Dialogic® Brooktrout® Bfv APIs

Reference Manual

September 2019 931-133-11

www.dialogic.com

Copyright and Legal Notice

Copyright © 1998-2019 Dialogic Corporation. All Rights Reserved. You may not reproduce this document in whole or in part without permission in writing from Dialogic Corporation at the address provided below.

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation and its affiliates or subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in certain safety-affecting situations. Please see *http://www.dialogic.com/company/terms-of-use.aspx* for more details.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at *www.dialogic.com*.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 3300 Boulevard de la Côte-Vertu, Suite 112, Montreal, Quebec, Canada H4R 1P8. *Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.*

Dialogic, Dialogic Pro, DialogicOne, Dialogic Buzz, Brooktrout, BorderNet, PowerMedia, PowerVille, PowerNova, ControlSwitch, I-Gate, Veraz, Cantata, TruFax, and NMS Communications, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation and its affiliates or subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 3300 Boulevard de la Côte-Vertu, Suite 112, Montreal, Quebec, Canada H4R 1P8. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

The names of actual companies and products mentioned herein are the trademarks of their respective owners.

Hardware Limited Warranty

Refer to the following Dialogic web site for information on hardware warranty information, which applies unless different terms have been agreed to in a signed agreement between yourself and Dialogic Corporation or its subsidiaries. The listed hardware warranty periods and terms are subject to change without notice. For purchases not made directly from Dialogic please contact your direct vendor in connection with the warranty period and terms that they offer.

http://www.dialogic.com/warranties

Contents

About this Publication 21

Related Documents 23
Operating System Support and Testing 23
Manual Conventions 23
Terminology 25
Updated Terminology 25
Getting Technical Support 27

Volume 1 – Administration, Management, and Configuration. . .28

About this Volume 28

Chapter 1 – Bfv API Overview 29

Using Structure Packing 31
Using Bfv API Function Argument Structures 31
Supported Arguments 32
Configuring Call Control 33
Bfv API Function Locator 34

Chapter 2 – Administration and Initialization 46

The BTLINE Structure 47
Function Summary 49
BfvCheckAddress 50

BfvCheckFacility 52
BfvLineAttach 54
BfvLineConfig 57
BfvLineDetach 60
BfvLineInfo 61
BfvLineReset 63
BfvLinesAvail 67
BfvModuleDeactivate 69
BfvModuleInfo 71
BfvSessionAttach 78
BfvSessionDetach 81
Macros 82
Low-Level Macros 90
Handling Alerts 92

Chapter 3 – Firmware 93

Function Summary 94
BfvFeatureSetDownload 95
BfvFeatureSetDownloadData 97
BfvFeatureSetQuery 99
BfvFirmwareDownload 101
BfvFirmwareDownloadData 105
BfvModuleConfigSpecsGet 109
Macros 111

Chapter 4 – Configuration 116

Configuration Files 117
Function Summary 118
BfvCallSWClearConns 119
BfvCallSWConnect 121
BFVCallSWConnectIP 127
BfvCallSWGetConns 142
BfvCallSWGetInfo 146
BfvNetworkConfigGet 149
BfvNetworkConfigSet 153
BfvNetworkQuery 158

BfvTelephGetInfo 163
BfvTelephReset 165
BfvTelephSave 167

Chapter 5 – Status and Monitoring 169

Function Summary 170
BfvBoardNotify 171
BfvBoardStateGet 178
BfvBoardStateSet 180
BfvBoardTemperatureGet 182
BfvBoardTemperatureThreshSet 184
BfvBoardTest 186
BfvIPCallControlNotify 190
BfvRtpEventControl 195
BfvRtpEventGet 198
BfvRtcpReportSend 206

Chapter 6 – Miscellaneous Functions 209

Function Summary 210
dll... 211
BfvGetVar 212
BfvLineAlert 215
BfvMemAllocFuncsSet 218
BfvRcvProcessPkt 221
BfvSetSingleVar 224
getopt 228

Chapter 7 – Debugging, Error Handling and Return Values 230

Structures and Return Values 232
Function Summary 233
BfvDebugFuncSet 234
BfvDebugInitData 236
BfvDebugModeSet 237
BfvDebugModeSetAdv 239
BfvErrorMessage 244
BfvHistoryClear 246

BfvHistoryClearModChan 248
BfvHistoryClearUnit 250
BfvHistoryDump 252
BfvHistoryDumpModChan 255
BfvHistoryDumpUnit 258
BfvLineDumpStructure 261
Macros 263
RES Structure Parameters 265

Volume 2 – Bfv-Level Call Control and Call Switching 266

About this Volume 266

Chapter 8 – Call Control Overview. 267

Bfv-Level Call Control 268
BSMI-Level Call Control 270

Chapter 9 – Bfv-Level Call Control 271

About Bfv API-Level Call Control 272
Bfv API High-Level Call Control Summary 273
Bfv API Low-Level Call Control Summary 274
Bfv API Protocol-Specific Call Control Function Summary 276
ISDN Services Call Control Summary 277
Call Control Configuration File 279
BfvCallAccept 280
BfvCallCtrlClose 283
BfvCallCtrlInit 284
BfvCallDisconnect 287
BfvCallHold 289
BfvCallReconfigureHostModule 291
BfvCallReject 293
BfvCallRetrieve 295
BfvCallRingDetect 297
BfvCallSendAlerting 301
BfvCallSetup 303
BfvCallSignalingStateMonitor 319
BfvCallSignalingStateSet 323

BfvCallStatus 326
BfvCallTransferComplete 329
BfvCallWaitForAccept 331
BfvCallWaitForAlerting 334
BfvCallWaitForComplete 337
BfvCallWaitForHold 345
BfvCallWaitForRelease 347
BfvCallWaitForRetrieve 350
BfvCallWaitForSetup 352
BfvCallWaitTransferComplete 359
BfvLineAnswer 362
BfvLineCCProtocolGet 365
BfvLineDialString 368
BfvLineOriginateCall 374
BfvLineTerminateCall 392
BfvLineTransfer 396
BfvLineTransferCancel 402
BfvLineTransferCapabilityQuery 404
BfvLineTransferComplete 406
BfvLineWaitForCall 408
BfvLoopCurrentDetectDisable 416
BfvLoopCurrentDetectEnable 418

Chapter 10 – Dialing Database Functions 420

Dialing Database Function Call Summary 422
BfvDialDBCheck 423
BfvDialDBList 425
BfvDialDBUpdate 428
BfvLineOrigCallDB 431

Chapter 11 – Data Structures 438

Low-Level Call Control (args_cc) 439
Functions Using the args_cc Structure 459
High-Level Call Control (args_telephone) 468
Macros 487

Volume 3 – Media Processing 488

About This Volume 488

Chapter 12 – Signal Generation and Detection 489

Signal Generation/Detection Function Summary 491
BfvCPGen 492
BfvCPGenAdv 494
BfvDataCP 498
BfvLineCallProgressDisable 502
BfvLineCallProgressEnable 504
BfvLineCallProgressProgram 509
BfvToneDetectDisable 513
BfvToneDetectEnable 515
BfvToneFlush 518
BfvToneGet 519
BfvTonePeek 521
BfvTonePlay 523
BfvTonePlayBeep 525
BfvToneUnget 528

Chapter 13 – Voice Play and Record 529

Voice Play and Record Function Summary 530
BfvPromptPlay 531
BfvSpeechEchoCancelControl 535
BfvSpeechModify 538
BfvSpeechPlay 541
BfvSpeechPlayData 544
BfvSpeechPlayFile 551
BfvSpeechPlayWave 557
BfvSpeechRecord 561
BfvSpeechRecordData 570
BfvSpeechRecordFile 580
BfvSpeechRecordWave 589
Macros 598

Chapter 14 – Infopkt File Functions 599

InfoPkt Function Summary 600
BfvInfopktClose 601
BfvInfopktFseek 603
BfvInfopktFtell 605
BfvInfopktGet 607
BfvInfopktOpen 609
BfvInfopktOpenMem 611
BfvInfopktPut 615
BfvInfopktUnget 617
BfvInfopktUser 619
BfvPromptClose 622
BfvPromptOpen 624

Volume 4 – Fax Processing 627

About this Volume 627

Chapter 15 – Fax Overview 628

Chapter 16 – Fax Functions 630

Fax Function Summary 631
BfvDataFSK 634
BfvFaxAbort 638
BfvFaxBegin 640
BfvFaxBeginRaw 647
BfvFaxBeginReceive 654
BfvFaxBeginSend 659
BfvFaxBeginSendRaw 662
BfvFaxBeginSendTiff 667
BfvFaxBeginTiff 670
BfvFaxDownloadFont 675
BfvFaxDownloadFontData 679
BfvFaxEndOfDocument 682
BfvFaxEndReception 684
BfvFaxGetRemoteInfo 686

BfvFaxHeader 688
BfvFaxNextPage 692
BfvFaxNextPageDCX 695
BfvFaxNextPageRaw 698
BfvFaxNextPageTiff 701
BfvFaxPageParams 704
BfvFaxPoll 706
BfvFaxRcvPageDCX 711
BfvFaxRcvPageTiff 713
BfvFaxReceive 716
BfvFaxReceiveData 721
BfvFaxReceiveFile 725
BfvFaxReceivePage 728
BfvFaxReceivePages 730
BfvFaxSend 732
BfvFaxSendData 736
BfvFaxSendFile 738
BfvFaxSendPage 741
BfvFaxSendPageDCX 744
BfvFaxSendPageTiff 746
BfvFaxSetLocalId 748
BfvFaxSetNSF 750
BfvFaxSetReceiveFmt 753
BfvFaxSetSubPwdSep 756
BfvFaxStripParams 759
BfvFaxT30Holdup 763
BfvFaxT30Params 769
BfvFaxT4TimerParams 773
BfvFaxWaitForTraining 776
Macros 778

Chapter 17 – TIFF-F Files Functions 781

TIFF-F Files Function Summary 782
BfvTiffClose 783
BfvTiffOpen 785
BfvTiffReadIFD 787
BfvTiffReadImage 790

BfvTiffReadRes 792
BfvTiffWriteIFD 794
BfvTiffWriteImage 797
BfvTiffWriteRes 799
Macros 801

Volume 5 – BSMI-Level Call Control and Call Switching 803

About this Volume 803

Chapter 18 – BOSTON Simple Message Interface (BSMI) 804

BSMI Installation 806
BSMI Function Summary 807
BsmiClearVtty 808
BsmiCloseAdapter 809
BsmiControlRead 810
BsmiControlWrite 812
BsmiLineAlert 813
BsmiModuleList 815
BsmiOpenAdapter 817
BsmiResetAdapter 819
BsmiSetVtty 821
BsmiVttyRead 822
BsmiVttyWrite 823
Error Return Values 824
Firmware Download 826
BSMI Use Examples 827
- Initialization and BSMI Message Sequence 827
- BRI Protocol Stack Initialization 830

Chapter 19 – BSMI General Message Structure 831

BSMI Message Naming Convention 832
BSMI Control Messages by Category 833
- Management Messages 833
- Call Control Messages 836
- ISDN Supplemental and Miscellaneous Messages 841

L4L3 Message Common Header 842

L3L4 Message Common Header 844
L4 Reference and Call Reference 846
L4 Reference Value 846
Call Reference Value 847
Relationship between L4 Reference and Call Reference 848
Logical Link ID or DLCI 849
Common Structures 850
Alerting and Connecting Data Message (IISDN_AL_CON_DATA) 851
Call ID (IISDN_CALL_ID) 854
Called Party (IISDN_CALLED_PARTY) 855
Calling Party (IISDN_CALLING_PARTY) 857
Cause Data (IISDN_CAUSE) 860
Connected Address (IISDN_CONNECTED_ADDRESS) 863
Information Element (IISDN_IE_STRUCT) 865
Progress Indication (IISDN_PROGRESS) 867
Q.933 DLCI Negotiation (IISDN_Q922_DLCI) 870
Redirecting Number (IISDN_REDIRECT_NUM) 871
User Info (IISDN_USER_INFO) 875

Chapter 20 – R2 Signaling Protocol with BSMI 876

Application to Stack (Host to Module) Messages 877
Numbering Conventions 878
Arguments 878
R2 Signaling L4L3 Messages 880
L4L3mALERTING_REQUEST 881
L4L3mCALL_PROCEEDING_REQUEST 883
L4L3mCALL_REQUEST 885
L4L3mCAS_CHAN_BLOCK 888
L4L3mCAS_CHAN_UNBLOCK 890
L4L3mCLEAR_REQUEST 892
L4L3mCOLLECT_DIGITS 894
L4L3mCONNECT_REQUEST 896
L4L3mDISABLE_CAS 898
L4L3mENABLE_CAS 900
L4L3mINFO_REQUEST 903
L4L3mREQ_ABCD_DATA 905
L4L3mSET_CAS_SIGNALING_BITS 906

Stack to Application (Module to Host) Messages 908
Normal Event Sequence 910
Inbound Calls 910
Outbound Calls 911
R2 Signaling L3L4 Messages 912
L3L4mALERTING 913
L3L4mCAS_CHAN_BLOCKED 914
L3L4mCAS_CHAN_UNBLOCKED 916
L3L4mCAS_SIGNALING_BIT_STATUS 917
L3L4mCAS_STATUS 919
L3L4mCLEAR_REQUEST 921
L3L4mCONN_ACK_IND 923
L3L4mCONNECT 924
L3L4mDISCONNECT 925
L3L4mERROR 927
L3L4mPRE_SEIZE 929
L3L4mSETUP_IND 930

Chapter 21 – LEC Protocols with BSMI 931

Application to Stack (Host to Module) Messages 932
Numbering Conventions 933
Arguments 934
LEC Signaling L4L3 Messages 935
L4L3mCALL_REQUEST 936
L4L3mCLEAR_REQUEST 939
L4L3mCOLLECT_DIGITS 941
L4L3mCONNECT_REQUEST 943
L4L3mDIAL 945
L4L3mDISABLE_CAS 948
L4L3mENABLE_CAS 950
L4L3mEND_DIAL 953
L4L3mFORCE_CONNECTION_REQUEST 955
L4L3mREQ_ABCD_DATA 957
L4L3mREQ_CONFIGURATION 958
L4L3mSET_CONFIGURATION 960
L4L3mTX_HOOKFLASH 962
L4L3mTX_WINK 964

Stack to Application (Module to Host) Messages 966
LEC Signaling L3L4 Messages 969
L3L4mABCD_SIGNAL_DATA 970
L3L4mACK_DOWNLOAD 971
L3L4mACK_UPLOAD 972
L3L4mALERTING 973
L3L4mCALLER_ID_DETECTED 974
L3L4mCAS_SIGNALING_BIT_STATUS 977
L3L4mCAS_STATUS 979
L3L4mCLEAR_REQUEST 980
L3L4mCONFIGURATION_STATUS 982
L3L4mCONN_ACK_IND 983
L3L4mCONNECT 984
L3L4mDISCONNECT 985
L3L4mEND_DIAL 987
L3L4mERROR 988
L3L4mHOOKFLASH 991
L3L4mLOOP_ON 992
L3L4mLOOP_REVERSAL 994
L3L4mPRE_SEIZE 995
L3L4mPROGRESS 996
L3L4mRING_STATUS 997
L3L4mRX_WINK 999
L3L4mSEIZE_COMP 1000
L3L4mSETUP_IND 1001
L3L4mSTATUS_IND 1002
L3L4mTX_HOOKFLASH_END 1003
L3L4mTXWINK_END 1004

Chapter 22 – Host to Module (L4L3m) Messages 1005

L4L3mALERTING_REQUEST 1006
L4L3mCALL_PROCEEDING_REQUEST 1008
L4L3mCALL_REQUEST 1012
L4L3mCLEAR_REQUEST 1020
L4L3mCONNECT_REQUEST 1023
L4L3mDISABLE_B_CHANNEL 1025
L4L3mDISABLE_PROTOCOL 1029

L4L3mENABLE_B_CHANNEL 1030
L4L3mENABLE_PROTOCOL 1033
L4L3mFACILITY_REQUEST 1052
L4L3mFEATURE_REQUEST 1055
L4L3mINFO_REQUEST 1059
L4L3mJATE_REDIAL 1061
L4L3mPROGRESS_REQUEST 1063
L4L3mREQ_BOARD_ID 1068
L4L3mREQ_LINE_STATUS 1069
L4L3mREQ_L2_STATS 1070
L4L3mREQ_PROTOCOL_STATUS 1072
L4L3mRESTART 1073
L4L3mSET_HARDWARE 1077
L4L3mSETUP_ACK_REQUEST 1078
L4L3mUNIVERSAL 1081
L4L3mUSER_INFO 1085

Chapter 23 – Module to Host (L3L4m) Messages 1086

L3L4mALERTING 1087
L3L4mANI 1088
L3L4mB_CHANNEL_STATUS 1090
L3L4mBILLING_STATUS 1093
L3L4mBOARD_ID 1094
L3L4mCALL_PROCEEDING 1096
L3L4mCALL_PROC_SENT 1097
L3L4mCLEAR_REQUEST 1098
L3L4mCLEAR_WITH_RESTART_REQUEST 1100
L3L4mCONNECT 1102
L3L4mCONN_ACK_IND 1103
L3L4mDISCONNECT 1104
L3L4mERROR 1106
L3L4mINFO_REQUEST 1111
L3L4mL2_STATS 1113
L3L4mLINE_STATUS 1116
L3L4mPROGRESS 1118
L3L4mPROTOCOL_STATUS 1119
L3L4mRAW_QDATA 1126

L3L4mRESTART 1128
L3L4mSETUP_IND 1129
L3L4mSTATUS_IND 1134
L3L4mUNIVERSAL 1135
L3L4mUSER_INFO 1137

Chapter 24 – B-Channel and D-Channel Maintenance 1138

B-Channel Maintenance 1139
ISDN Messages for B-Channel Maintenance 1140
BSMI Messages For B-Channel Maintenance 1140
Maintenance Procedures 1142
D-Channel Maintenance 1147
BSMI Messages for D-Channel 1147

Volume 6 – Appendices. 1149

About this Volume 1149

Appendix A – Configuration Files 1150

User-Defined Configuration File 1152
Call Control Configuration File 1170
Call Control Configuration File Format 1171
Global Options 1176
Global Module Parameters 1178
Clock Configuration Parameters 1186
Port Configuration Parameters 1190
Specific Parameters for Port Configuration 1193
Internet Protocol (IP) Call Control Configuration Parameters 1245
Examples of PSTN Call Control (callctrl.cfg) Files 1296
Examples of IP Call Control Configuration File 1307
Sample Configuration Files 1318
Routing Table Configuration File 1322
Routing Table Configuration File Format 1323
Routing Rule Parameters 1325
Examples of Routing Table Configuration Files 1328
SRTP Configuration File 1331
TLS Configuration File 1334

Parameters for Technical Support Purposes 1337

Appendix B – Bfv API Structures 1342

Address Structure 1343
Result Structures 1344
RES Structure Parameters 1348
CALL_RES Structure Parameters 1352
INFO_RES Structure Parameters 1356
PAGE_RES Structure Parameters 1357
FAX_RES Structure Parameters 1362
DCS and DIS/DTC Info Structures 1367

Appendix C – Hangup Codes 1371

Call Placement Codes 1373
Transmit Phase A Codes 1373
Transmit Phase B Codes 1374
Transmit Phase D Codes 1376
Receive Phase B Codes 1380
Receive Phase D Codes 1382
Phase C Codes 1383
Miscellaneous Codes 1384
Bfv API-Created Codes 1385

Appendix D – BSMI and ISDN Cause Codes 1386

Defining BSMI Cause Codes 1386
Defining ISDN Cause Codes 1390

Appendix E – Infopkt Parameter Values 1398

Voice Infopkt Parameters 1400
End-of-Speech Parameter Infopkt 1400
Prompt Map Infopkt 1401
Speech Parameters Infopkt 1402
Fax Infopkt Parameters 1404
ASCII Strip Infopkt 1404
Document Parameters Infopkt 1406

Enhanced Fax Format Page Infopkt 1408
Fax Header Parameters Infopkt 1409
G3 Strip Infopkt 1410
Page Parameters Infopkt 1412
T.30 Parameters Infopkt 1414
Beginning of Page Infopkt 1417

Appendix F – Call Progress Notes 1418

Processing Call Progress Signals 1419
Adapting to International Specs 1420
Reporting Call Progress Results 1421
Intermediate Results 1421
Final Results 1421
Initiating Call Progress 1423
BfvLineCallProgressEnable 1423
BfvLineOriginateCall 1424
Setting the Call Progress Mode 1425
Voice Mode 1425
Fax Mode 1425
Raw Mode 1425
Special Call Progress Features 1426
Sending CNG 1426
Call Progress Analysis During Dialing 1426
Call Progress Signals 1427
Special Information Tones 1434
Custom Call Progress Results 1436
Final Call Progress Results 1438

Appendix G – Country-Specific Parameter Files 1439

BT_CPARM.CFG Parameter File 1441
Using Dialing Database Functions and Dialing Parameters 1448
Country-Specific Dialing Requirements 1450
Australia 1450
Canada 1451
Czech Republic 1451
Denmark 1452

European Community (Boards Approved to TBR 4) 1452
France 1452
Germany 1453
Hong Kong 1453
Ireland 1453
Israel 1454
Italy 1454
Japan 1454
Malaysia 1455
Netherlands 1455
New Zealand 1455
Norway 1456
Singapore 1456
Spain 1456
Switzerland 1457
Turkey 1457
United Kingdom 1457
United States 1458
Examples of R2 Parameter Files 1459
Argentina R2 Parameter File 1459
Brazil R2 Parameter File 1462
China R2 Parameter File 1469
Korea R2 Parameter File 1472
Mexico R2 Parameter File 1475

Appendix H – Unsupported Functionality 1478

Appendix I – SR140 Security Capabilities 1483

Secure RTP (SRTP) 1484
SIP over TLS 1485
FIPS 1493

Index 1494

About this Publication

This publication is made up of the following volumes:

- *Volume 1, Administration, Management, and Configuration, Bfv API Reference Manual* provides information about the following Dialogic® Brooktrout® Bfv API components:
  - Line Administration and Initialization functions
  - Firmware, Configuration, and Status functions
  - Error Management and Miscellaneous functions
- *Volume 2, Bfv-Level Call Control and Call Switching, Bfv API Reference Manual* provides information about the following Bfv API components:
  - Bfv API-level Call Control functions
  - Dialing Database functions
  - Call Control data structures and macro
- *Volume 3, Media Processing, Bfv API Reference Manual* provides information about the following Bfv API components:
  - Signal Generation and Detection functions
  - Voice Play and Record functions
  - Infopkt file functions
- *Volume 4, Fax Processing, Bfv API Reference Manual* provides information about the following Bfv API components:
  - Fax functions and macros
  - TIFF-F files functions and macros

- *Volume 5, BSMI-Level Call Control and Call Switching, Bfv API Reference Manual* provides information about the following BOSTON Simple Message Interface (BSMI) Bfv API components:
  - BSMI-level Call Control functions
  - Message structure
  - R2 Signaling Protocol messages
  - Local Exchange Carrier (LEC) Protocol messages
  - Host to Module and Module to Host messages
  - B-Channel and D-Channel Maintenance procedures
- *Volume 6, Appendices, Bfv API Reference Manual* is a grouping of appendices that relate to the reference material in Volumes 1 through 5, including:
  - Configuration files
  - Bfv API structures
  - Hangup codes
  - BSMI and ISDN cause codes
  - Infopkt parameter values
  - Call Progress notes
  - Country-specific parameter files
  - Deprecated and unsupported functionality
  - SR140 security capabilities

Related Documents

- The *Dialogic® Brooktrout® Fax Products SDK Installation and Configuration Guide* explains how to install the software (firmware, Bfv API, and driver for the Dialogic® Brooktrout® TR1034/TruFax® and Dialogic® Brooktrout® SR140 Fax Software) on your host system. It also describes how to configure the driver, configure call control, and download the firmware to a module.
- The *Dialogic® Brooktrout® Fax Products SDK Developer Guide* describes the Bfv API and gives information about Call Transfer, Automatic Speech Recognition, BSMI, and how to package Dialogic® Brooktrout® fax software for your customers.

Operating System Support and Testing

For the latest list of operating systems supported on your product see the applicable *Release Notes*.

Manual Conventions

This manual uses the following conventions:

- *Italics* denote the names of variables in the prototype of a function and file names, directory names, and program names within the general text.
- The **`Courier`** font in bold indicates a command sequence entered by the user at the system prompt, for example:

  `cd /Brooktrout/boston/bfv.api`
- The `Courier` font not bolded indicates system output, for example:

  `C:>Files installed.`
- The `Courier` font also denotes programming code, such as C, C++, Visual Basic, and TSL. Programming code appears in program examples.

- *Bold* indicates the data type of the prototype of a function, Bfv API functions, dialog boxes, dialog box controls, windows, and menu items.
- Square brackets [ ] indicate that the information to be typed is optional.
- Angle brackets < > indicate that you must supply a value with the parameter.

The Caution icon is used to indicate an action that could cause harm to the software or hardware.

The Warning icon is used to indicate an action that could cause harm to the user.

Terminology

Updated Terminology

The current version of this document includes terminology that differs from previous versions. Please note the changes below:

Former Terminology	Replaced with...
Host-based fax	Dialogic® Brooktrout® SR140 Fax Software or Brooktrout SR140 Fax Software or SR140 Software or SR140
Virtual modules	
Virtual boards	
Software modules	
VoIP modules	
SR140 virtual modules	
TR1000 Series SDK	Dialogic® Brooktrout® SDK
TR1000 Series Product	Dialogic® Brooktrout® Fax Board or Brooktrout fax board or board
TR1000 Series Module	
TR1000 Series Board	
Brooktrout System Software	Dialogic® Brooktrout® Runtime Software

Dialogic® Brooktrout® TR1034 Fax Board Terminology

The Dialogic® Brooktrout® TR1034 Fax Board is also referred to herein by one or more of the following terms, or like terms including “TR1034”:

- Brooktrout TR1034 Fax Board
- Brooktrout TR1034 Board
- TR1034 Fax Board
- TR1034 Board

Getting Technical Support

Dialogic provides technical services and support for customers who have purchased hardware or software products from Dialogic. If you purchased products from a reseller, please contact that reseller for technical support.

To obtain technical support, please use the web site below:

www.dialogic.com/support

Volume 1 - Administration, Management, and Configuration

About this Volume

- *Volume 1, Administration, Management, and Configuration,* provides information about the following Dialogic® Brooktrout® Bfv API components:
  - Line Administration and Initialization functions
  - Firmware, Configuration, and Status functions
  - Error Management and Miscellaneous functions

1 - Bfv API Overview

This chapter provides an introduction to the Bfv API, briefly describing the interface and the functions.

The functions described in this manual define the interface between a developer application code and the available features provided by the Dialogic® Brooktrout® hardware, driver, and firmware components. This interface enables an application programmer to develop broad applications with voice, fax, DTMF, and call control elements with relative simplicity.

Note: To find Bfv API function descriptions within the Bfv API Volume Set, see the *Bfv API Function Locator on page 34*.

The Dialogic® Brooktrout® products in the Brooktrout SDK store their *include* files in several directories below the directory *<install_root>/boston*. For example, the core Bfv API files are stored in *boston/bfv.api/inc*. See your *Bfv API SDK Installation and Configuration Guide* for the directory locations and contents.

Note: The *install_root* for Linux systems is */usr/sys/brooktrout* and the root for Windows operating systems is *C:\brooktrout*.

The following table shows the libraries and driver version locations for the software products.

Bfv API library	*boston/bfv.api/inc/btlib.h*
Bfv API library version	*boston/bfv.api/inc/apiver.h*
Audio Conferencing API (ACC) library	*boston/acc.api/inc/acclib.h*
BOSTON Simple Message Interface (BSMI) library	*boston/bsmi.api/inc/bsmilib.h*, *boston/bsmi.api/inc/iisdn.h*
Call Control library	*boston/bfv.api/inc/ecclib.h*
Driver version	*boston/driver/inc/millver.h*

When writing applications for DLL environments, see the **BT_API_SET_VER** macro described in *Volume 1, Chapter 2* for information about drop-in replacement of future DLL API libraries.

Some platforms support the use of multiple threads. In such an environment, it is never permissible to access the same line pointer from more than one thread simultaneously. Doing so might cause unpredictable behavior.

Defining the BCONST symbol (on the compiler command line or before including *btlib.h*) causes the compiler to apply the ANSI const modifier to some of the pointer arguments of Bfv API function prototypes or argument structure fields. Using this optional feature enables applications to use the const modifier without generating pseudoconflicts when they call Bfv API functions.

To use the Bfv API with C++ applications, change the inclusion of *btlib.h* to use the extern "C" construct as follows:

```
extern "C"
{
#include "btlib.h"
}
```

Using Structure Packing

Many compilers allow users to specify structure packing, either on the compiler command line or within a source file using **`#pragma pack.`**

You should verify that the default packing level be in effect when Dialogic® Brooktrout® header files are included to help ensure that there are no conflicts between applications being compiled and the Bfv API libraries.

The recommendation also holds true for any system-supplied header files.

Using Bfv API Function Argument Structures

Most Bfv API functions use an argument structure. The argument structure is declared in an application, and a pointer to it is passed to the function. The argument structure type is named *args_...*; for example, *struct args_fax*. The same argument structure type is used for functions that are related or in the same category.

Structure fields contained within the argument structure are used for input and/or output. Each function that uses an argument structure has marked the fields that are used for each purpose. Not all fields are used by all functions taking any particular argument structure type.

Note: You *must* clear each argument structure (fill with 0s) before its first use. This can be accomplished using the memset function or the supplied BT_ZERO macro.

Use of memset may require inclusion of an appropriate header file such as <*string.h*>. Once you clear an argument structure, it can be used for multiple calls to Bfv API functions that accept this argument structure type. Be sure that any argument structure fields that might have been modified by a previous call (listed as Modified fields) are set to appropriate values, usually 0, before the next call. It is always safe to clear an argument structure before using it.

The Bfv API and driver for the Brooktrout SDK are written assuming ANSI compilers and 32-bit integers/addressing. Non-ANSI compilers or 16-bit platforms are not supported.

Supported Arguments

Not all members of the argument structures that are used to pass arguments to the Bfv API function calls are supported. An argument in the header file is only supported if the documentation defines the argument. Similarly, the enums or defines in a header file are only supported if the documentation defines the values. The reasons include:

- The Bfv API often defines the interface in advance of the existence of certain functionality.
- The same structure can be used for many Bfv API functions.
- Items were used in the past but are no longer in use or remain only to maintain compatibility.

Configuring Call Control

To use the call control functionality provided in the Brooktrout SDK, create a call control configuration file (*callctrl.cfg*) and use applicable call control functions as follows:

- For Bfv API-level call control functions, use *Volume 2, Bfv-Level Call Control and Call Switching*
- For BSMI-level call control functions, use *Volume 5, BSMI-Level Call Control and Call Switching*

The call control configuration file is an ASCII file that contains general configuration parameters for all telephony hardware modules and static telephony connections to be formed for all modules. This file replaces the *teleph.cfg* and *ecc.cfg* files that Dialogic no longer supports.

Removing support for *teleph.cfg* and *ecc.cfg* files also affects the following:

- The *teleph* parameter in the user-defined configuration file

  Delete this parameter and set the location of your call control configuration file as the value for the *call_control* parameter.
- The symbol `NO_ECC` as a compiling option

  The Bfv API ignores this compiling option.
- The header files and libraries

  The *BfvTelephConfig* and *BfvTelephConfigData* functions and the *callctrl.h* file are deleted from the Bfv header files and libraries.

For all supported operating systems, create a call control configuration file or edit one of the sample files supplied with the Brooktrout SDK. Dialogic also provides the *Dialogic® Brooktrout® Configuration Tool*, a graphical user interface (GUI) program, as an option for Windows users to create or modify a configuration file for the call control software.

See *Volume 6, Appendix A* and your installation and configuration guide for more information about the new call control and associated parameters.

Bfv API Function Locator

The table below provides the location and a high-level description of each function within the Bfv API Reference Volume set.

Function	Location	Purpose
dll...	*Vol 1: 211*	Calls the standard C library function that matches the _dll_...function with the arguments provided by using the runtime library linked with the dll.
BfvGetVar	*Vol 1: 212*	Waits for notification of async activity on another line pointer.
BfvBoardNotify	*Vol 1: 171*	Turns module level notification on or off and sets up an optional callback function.
BfvBoardStateGet	*Vol 1: 178*	Retrieves the current state of a module (Board Status LED).
BfvBoardStateSet	*Vol 1: 180*	Sets the current state of a module (Board Status LED).
BfvBoardTemperatureGet	*Vol 1: 182*	Retrieves the current temperature of a module.
BfvBoardTemperatureThreshSet	*Vol 1: 184*	Sets the temperature threshold of a module for the purpose of module event notification.
BfvBoardTest	*Vol 1: 186*	Initiates a series of self-tests on the module and reports the results as they occur through an optional callback function.
BfvCallAccept	*Vol 2: 280*	Starts answering an incoming telephone call.
BfvCallCtrlClose	*Vol 2: 283*	Disables signaling on all the ISDN spans in the system and shuts down the call control library.
BfvCallCtrlInit	*Vol 2: 284*	Initializes the call control runtime environment.
BfvCallDisconnect	*Vol 2: 287*	Starts the process of terminating a telephone call.
BfvCallHold	*Vol 2: 289*	Places the Bfv API in the hold state.
BfvCallReconfigureHostModule	*Vol 2: 291*	Forces the specified third party call control stack (host module) to reread its call control configuration file while the system is running.

Function	Location	Purpose
BfvCallReject	*Vol 2: 293*	Rejects an incoming telephone call on line types or protocols that allow call rejection.
BfvCallRetrieve	*Vol 2: 295*	Takes the Bfv API out of the hold state.
BfvCallRingDetect	*Vol 2: 297*	Turns detection of ring signals on or off and determines the type of detection for notification of incoming calls.
BfvCallSendAlerting	*Vol 2: 301*	Sends an ALERTING message to the remote end after detecting an incoming call.
BfvCallSetup	*Vol 2: 303*	Starts the process of dialing an outgoing telephone call or transferring a call.
BfvCallSignalingStateMonitor	*Vol 2: 319*	Turns inbound call signaling state monitoring on or off and sets up an optional callback function.
BfvCallSignalingStateSet	*Vol 2: 323*	Sets the outbound call signaling state to one value for a specified time and then to a second value.
BfvCallStatus	*Vol 2: 326*	Retrieves the channel's current call state.
BfvCallSWClearConns	*Vol 1: 119*	Clears all call switching connections on the current module.
BfvCallSWConnect	*Vol 1: 121*	Forms a connection between specified source and destination telephony resources.
BfvCallSWGetConns	*Vol 1: 142*	Retrieves and returns information about established call switching connections on the current module.
BfvCallSWGetInfo	*Vol 1: 146*	Retrieves and returns information about the connectable port classes and units.
BfvCallTransferComplete	*Vol 2: 329*	Completes a call transfer without waiting for the transfer process to complete.
BfvCallWaitForAccept	*Vol 2: 331*	Finishes the process of answering an incoming telephone call.
BfvCallWaitForAlerting	*Vol 2: 334*	Waits for an outgoing call to finish establishing or dialing.
BfvCallWaitForComplete	*Vol 2: 337*	Waits for the outgoing telephone call to finish.
BfvCallWaitForHold	*Vol 2: 345*	Finishes the process of diverting an incoming call on a digital port using the QSIG protocol.

Function	Location	Purpose
BfvCallWaitForHold	*Vol 2: 345*	Waits for the Bfv API to finish transitioning to the hold state.
BfvCallWaitForRelease	*Vol 2: 347*	Waits for the termination of a telephone call to finish.
BfvCallWaitForRetrieve	*Vol 2: 350*	Waits for the Bfv API to finish transitioning out of the hold state.
BfvCallWaitForSetup	*Vol 2: 352*	Waits for an incoming call, and returns all available information about the call to the application.
BfvCallWaitTransferComplete	*Vol 2: 359*	Waits for the line to complete the call transfer command.
BfvCheckAddress	*Vol 1: 50*	Checks for the existence of an address.
BfvCheckFacility	*Vol 1: 52*	Checks for the existence of the specified facility on the destination module and channel.
BfvConferenceAdminGetById	*Vol 3: 615*	Determines if the conference ID is in use or reserves it.
BfvConferenceAdminInitialize	*Vol 3: 617*	Initializes a system-wide conference administrator.
BfvConferenceAdminModify	*Vol 3: 620*	Modifies system-wide conference parameters for the conference administrator.
BfvConferenceAdminQuery	*Vol 3: 622*	Collects the current parameters and state values for the system-wide conference, including all conference IDs.
BfvConferenceAdminTerminate	*Vol 3: 625*	Terminates the conference administrator in the system and frees all administrative resources.
BfvConferenceBegin	*Vol 3: 627*	Initializes a new conference and creates a new conference object.
BfvConferenceDebugFuncSet	*Vol 3: 632*	Sets up a printing function to use with ACC debug mode output.
BfvConferenceDebugModeSet	*Vol 3: 634*	Enables ACC debug mode that prints commands, data, responses, and other status messages to the standard output or another location.
BfvConferenceDspRole	*Vol 3: 636*	Sets the role and other parameters for a conference DSP channel.
BfvConferenceEnd	*Vol 3: 640*	Ends one conferences in the system and frees all conference-related resources.

Function	Location	Purpose
BfvConferenceModify	*Vol 3: 642*	Changes conference parameters.
BfvConferencePartyAdd	*Vol 3: 647*	Adds a new party to an existing conference.
BfvConferencePartyDrop	*Vol 3: 653*	Removes a party from an existing conference.
BfvConferencePartyModify	*Vol 3: 655*	Changes the properties of an existing conference member.
BfvConferencePartyQuery	*Vol 3: 659*	Retrieves the properties of a conference member.
BfvConferenceQuery	*Vol 3: 663*	Provides information about the conference and its participants in a dynamically allocated array of conference party data structures.
BfvConferenceQueryFree	*Vol 3: 668*	Frees the array of internally allocated structures used to provide information about the conference and its members.
BfvCPGen	*Vol 3: 492*	Generates call progress signals.
BfvCPGenAdv	*Vol 3: 494*	Generates call progress signals and other tone patterns.
BfvDataCP	*Vol 3: 498*	Retrieves the next call progress code.
BfvDataFSK	*Vol 4: 634*	Fills an FSK buffer with FSK data for debugging aid.
BfvDebugFuncSet	*Vol 1: 234*	Sets up a function to use with Bfv API debug mode that directs debug output to an alternate destination.
BfvDebugInitData	*Vol 1: 236*	Recreates name tables used for Bfv API debug mode and the *dh* program, based on command set header files found in a specified directory.
BfvDebugModeSet	*Vol 1: 237*	Enables debug mode, so the Bfv API prints commands, data, interrupts, and status messages to the standard output or alternate device.
BfvDebugModeSetAdv	*Vol 1: 239*	Allows the application to configure Bfv API debugging features within the application, enabling the user to control debugging in a remote application.
BfvDialDBCheck	*Vol 2: 423*	Checks the specified dialing database for the specified telephone number, and returns the amount of time the application must wait before dialing the telephone number.

Function	Location	Purpose
BfvDialDBList	*Vol 2: 425*	Enables the application to read the contents of the specified dialing database.
BfvDialDBUpdate	*Vol 2: 428*	Updates the specified dialing database with the results of the most recent call to the specified telephone number.
BfvErrorMessage	*Vol 1: 244*	Returns error message strings corresponding to Bfv API errors returned in RES structures.
BfvFaxAbort	*Vol 4: 638*	Aborts a fax transmission or reception cleanly when possible.
BfvFaxBegin	*Vol 4: 640*	Initiates fax transmit or receive for infopkt streams. Handles all variations of polling.
BfvFaxBeginRaw	*Vol 4: 647*	Initiates fax transmit or receive for raw fax data. Handles all variations of polling.
BfvFaxBeginReceive	*Vol 4: 654*	Sets parameters and instructs the channel to receive.
BfvFaxBeginSend	*Vol 4: 659*	Begins fax transmission using an infopkt stream.
BfvFaxBeginSendRaw	*Vol 4: 662*	Initiates fax transmit for raw fax data.
BfvFaxBeginSendTiff	*Vol 4: 667*	Begins transmission using a TIFF-F file.
BfvFaxBeginTiff	*Vol 4: 670*	Initiates fax transmit or receive for TIFF-F files. Handles all variations of polling.
BfvFaxDownloadFont	*Vol 4: 675*	Loads a .fz8 font file to the channel as the specified font.
BfvFaxDownloadFontData	*Vol 4: 621*	Downloads a supplied font from the buffer to the channel as the specified numbered font.
BfvFaxEndOfDocument	*Vol 4: 682*	Sends an end-of-page with no more pages to follow.
BfvFaxEndReception	*Vol 4: 684*	Waits for completion of the T.30 confirmation handshaking sequence.
BfvFaxGetRemoteInfo	*Vol 4: 686*	Waits for and reports ID, DIS/DCS and NSF/NSS data.
BfvFaxHeader	*Vol 4: 688*	Sets up headers or footers on all subsequent pages in a fax transmission.
BfvFaxNextPage	*Vol 4: 692*	Sends an end-of-page and new page setup, if appropriate, for use with infopkt streams.
BfvFaxNextPageDCX	*Vol 4: 695*	Sends an end-of-page and new page setup, if appropriate, for use with DCX pages.

Function	Location	Purpose
BfvFaxNextPageRaw	*Vol 4: 698*	Sends an end-of-page and new page setup, if appropriate, for raw (noninfopkt-formatted) fax data.
BfvFaxNextPageTiff	*Vol 4: 701*	Sends an end-of-page and new page setup, if appropriate, for use with TIFF-F files.
BfvFaxPageParams	*Vol 4: 704*	Sets the page parameters for subsequent pages of data.
BfvFaxPoll	*Vol 4: 706*	High-level function that sends and/or receives faxes using infopkt streams.
BfvFaxRcvPageDCX	*Vol 4: 711*	Receives a fax page to a DCX file.
BfvFaxRcvPageTiff	*Vol 4: 713*	Receives a fax page to a TIFF-F file.
BfvFaxReceive	*Vol 4: 716*	High-level function that receives faxes using infopkt streams.
BfvFaxReceiveData	*Vol 4: 721*	Receives raw fax data into a user-supplied buffer.
BfvFaxReceiveFile	*Vol 4: 725*	Receives a raw fax page to a file.
BfvFaxReceivePage	*Vol 4: 728*	Receives a fax page to an infopkt stream.
BfvFaxReceivePages	*Vol 4: 730*	Receives multiple pages of fax data to an infopkt stream.
BfvFaxSend	*Vol 4: 732*	High-level function that sends faxes using infopkt streams.
BfvFaxSendData	*Vol 4: 736*	Sends raw fax data from a user-supplied buffer.
BfvFaxSendFile	*Vol 4: 738*	Sends a noninfopkt-formatted fax page from a file.
BfvFaxSendPage	*Vol 4: 741*	Sends an entire page from the infopkt stream to the driver buffer. Looks for an EOF or new page type infopkt before returning.
BfvFaxSendPageDCX	*Vol 4: 744*	Sends a fax page from a DCX file.
BfvFaxSendPageTiff	*Vol 4: 746*	Sends a fax page from a TIFF-F file.
BfvFaxSetLocalId	*Vol 4: 748*	Sets the local ID to a specified string.
BfvFaxSetNSF	*Vol 4: 750*	Sets up NSF, NSC, and NSS messages for transmission to the remote host.
BfvFaxSetReceiveFmt	*Vol 4: 753*	Sets the format of the received data.
BfvFaxSetSubPwdSep	*Vol 4: 756*	Sets up a SUB, PWD, or SEP FSK message to send to the remote host.

Function	Location	Purpose
BfvFaxStripParams	*Vol 4: 759*	Separates different strips of data and sets the strip parameters.
BfvFaxT30Holdup	*Vol 4: 763*	Causes the channel to wait during T.30 negotiations and calls a user-supplied function.
BfvFaxT30Params	*Vol 4: 769*	Sets the T.30 parameters for transmission.
BfvFaxWaitForTraining	*Vol 4: 776*	Reports when training is complete or `turn_around` is indicated.
BfvFeatureSetDownload	*Vol 1: 95*	Downloads feature set data supplied in a specified file.
BfvFeatureSetDownloadData	*Vol 1: 97*	Downloads feature set data supplied in a specified user buffer.
BfvFeatureSetQuery	*Vol 1: 99*	Queries the feature data currently stored in the module's feature set hardware.
BfvFirmwareDownload	*Vol 1: 101*	Downloads firmware to a module from a file.
BfvFirmwareDownloadData	*Vol 1: 105*	Downloads firmware to a module from a buffer.
BfvGetVar	*Vol 1: 212*	Requests the value of a specified facility firmware variable.
BfvHistoryClear	*Vol 1: 246*	Clears the driver's history buffer.
BfvHistoryClearModChan	*Vol 1: 248*	Clears the contents of the driver's history buffers for the specified module and channel number.
BfvHistoryClearUnit	*Vol 1: 250*	Clears the contents of the driver's history buffers on the channel specified by the channel number.
BfvHistoryDump	*Vol 1: 252*	Dumps the driver's history buffer to the specified open file.
BfvHistoryDumpModChan	*Vol 1: 255*	Dumps the contents of the driver's history buffer for the specified module and channel number to the specified open file.
BfvHistoryDumpUnit	*Vol 1: 258*	Dumps the contents of the driver's history buffer specified by the channel number to the specified open file.
BfvInfopktClose	*Vol 3: 601*	Closes the current infopkt file and frees all associated structure memory.
BfvInfopktFseek	*Vol 3: 603*	Searches to a specified offset in a file relative to a specified origin.

Function	Location	Purpose
BfvInfopktFtell	*Vol 3: 605*	Retrieves the position of the pointer within the specified infopkt stream file.
BfvInfopktGet	*Vol 3: 607*	Reads one infopkt from the stream. Can follow indirect infopkts if specified.
BfvInfopktOpen	*Vol 3: 609*	Opens the infopkt stream-formatted disk file.
BfvInfopktOpenMem	*Vol 3: 611*	Opens an infopkt stream associated with a user-supplied memory buffer instead of a file.
BfvInfopktPut	*Vol 3: 615*	Writes one infopkt to the infopkt stream file.
BfvInfopktUnget	*Vol 3: 617*	Replaces last infopkt in the infopkt stream so it is available for the next request.
BfvInfopktUser	*Vol 3: 619*	Sets up a user-supplied function to handle user-defined infopkts.
BfvLineAlert	*Vol 1: 215*	Interrupts an active channel for another use by suspending, but not killing, the interrupted process or thread.
BfvLineAnswer	*Vol 2: 362*	Answers an incoming call and sets the line state to CONNECTED.
BfvLineAttach	*Vol 1: 54*	Opens the given channel.
BfvLineCallProgressDisable	*Vol 3: 502*	Turns call progress off.
BfvLineCallProgressEnable	*Vol 3: 504*	Turns call progress on in one of three modes.
BfvLineCallProgressProgram	*Vol 3: 509*	Programs frequency and cadence analysis parameters for use during call progress monitoring.
BfvLineCCProtocolGet	*Vol 2: 365*	Retrieves the protocol assigned to the module.
BfvLineConfig	*Vol 1: 57*	Permits channel configuration via data buffers instead of a configuration file.
BfvLineDetach	*Vol 1: 60*	Closes, hangs up, and resets the line.
BfvLineDialString	*Vol 2: 368*	Places the line in an OFF_HOOK state, dials the digits specified, and returns after dialing the last digit.
BfvLineOriginateCall	*Vol 2: 374*	Starts to divert an incoming call and waits for the process to complete on a digital line using the QSIG protocol.

Function	Location	Purpose
BfvLineDumpStructure	*Vol 1: 261*	Dumps the contents of the BTLINE structure and configuration structures to the specified open file.
BfvLineInfo	*Vol 1: 61*	Returns the module type and channel address for a given channel from the driver.
BfvLineOrigCallDB	*Vol 2: 431*	Combines the functionality of *BfvDialDBCheck*, *BfvLineOriginateCall*, and *BfvDialDBUpdate*.
BfvLineOriginateCall	*Vol 2: 374*	Places a phone call on an outgoing line.
BfvLineReset	*Vol 1: 63*	Resets the channel and the state of the BTLINE structure.
BfvLinesAvail	*Vol 1: 67*	Returns the number of enabled channels.
BfvLineTerminateCall	*Vol 2: 392*	Hangs up a call and completes the disconnect process.
BfvLineTransfer	*Vol 2: 396*	Automatically transfers an incoming call from the called party to the dialed transfer number, or returns control to the application so that it can determine whether to complete or cancel the transfer.
BfvLineTransferCancel	*Vol 2: 402*	Ends a previously initiated call transfer and retrieves the original calling party.
BfvLineTransferCapabilityQuery	*Vol 2: 404*	Queries a channel's transfer capability and provides the application with information about pairs of lines available to perform a two B-channel call transfer.
BfvLineTransferComplete	*Vol 2: 406*	Completes the call transfer connection for a previously initiated call transfer.
BfvLineWaitForCall	*Vol 2: 408*	Waits for an incoming call.
BfvLoopCurrentDetectDisable	*Vol 2: 416*	Turns off loop current detection.
BfvLoopCurrentDetectEnable	*Vol 2: 418*	Turns on loop current detection on.
BfvMemAllocFuncsSet	*Vol 1: 218*	Replaces Bfv API functions that dynamically allocate and free memory with functions the application provides to do the same.
BfvModuleConfigSpecsGet	*Vol 1: 109*	Gets information about a module's possible firmware configuration options, as reported by the firmware.
BfvModuleDeactivate	*Vol 1: 69*	Deactivates a hardware module, marking it as dead.

Function	Location	Purpose
BfvModuleInfo	*Vol 1: 71*	Gets information about a module.
BfvNetworkConfigGet	*Vol 1: 149*	Returns network configuration information about a single specified interface or trunk.
args.unit	*Vol 1: 154*	Performs network configuration on a collection of interfaces and trunks.
BfvNetworkQuery	*Vol 1: 158*	Retrieves statistics about the specified Ethernet unit.
BfvPromptClose	*Vol 3: 622*	Closes the current prompt file and frees its associated memory.
BfvPromptOpen	*Vol 3: 624*	Opens a prompt file, a specialized infopkt file.
BfvPromptPlay	*Vol 3: 531*	Plays phrases from a prompt file.
BfvRcvProcessPkt	*Vol 1: 221*	Receives a packet, and performs internal Bfv API processing of all commands contained within the packet.
BfvSessionAttach	*Vol 1: 78*	Creates a session for communicating with any facilities on a specified channel located on a specified module and machine.
BfvSessionDetach	*Vol 1: 81*	Closes a specified channel and frees the BTLINE structure.
BfvSetSingleVar	*Vol 1: 224*	Attempts to send a SET command to set a single variable.
BfvSpeechEchoCancelControl	*Vol 3: 535*	Allows enabling, disabling, or resetting echo cancellation.
BfvSpeechModify	*Vol 3: 538*	Enables an application to modify the volume (gain) and rate of a speech playback while it is in progress.
BfvSpeechPlay	*Vol 3: 541*	Plays speech from an infopkt stream.
BfvSpeechPlayData	*Vol 3: 544*	Plays raw speech data from a data buffer.
BfvSpeechPlayFile	*Vol 3: 551*	Plays raw speech data from a file.
BfvSpeechPlayWave	*Vol 3: 557*	Plays speech from a wave file.
BfvSpeechRecord	*Vol 3: 561*	Retrieves the summation group number assigned to a channel.
BfvSpeechRecord	*Vol 3: 561*	Records speech in infopkt format.
BfvSpeechRecordData	*Vol 3: 570*	Records raw speech data into the specified buffer using the specified speech parameters.

Function	Location	Purpose
BfvSpeechRecordFile	*Vol 3: 580*	Records raw speech data into the specified file using the specified speech parameters.
BfvSpeechRecordWave	*Vol 3: 589*	Records speech into the specified wave (*.wav*) file using the specified speech parameters.
BfvTelephConfig	*Vol 6: Appendix H*	Reads a telephony configuration file, configures all telephony hardware units for the current module, and sets up telephony connections. (No longer supported)
BfvTelephConfigData	*Vol 6: Appendix H*	Configures a specific telephony hardware unit with specified parameters. (No longer supported)
BfvTelephGetInfo	*Vol 1: 163*	Retrieves and returns information about the telephony hardware units on the current module and their available port types.
BfvTelephReset	*Vol 1: 165*	Modifies the telephony state on the current module to permit performing high level configuration again using *BfvCallCtrlInit*.
BfvTelephSave	*Vol 1: 167*	Saves already configured telephony parameters to non-volatile memory (NVRAM) on the current module.
BfvTiffClose	*Vol 4: 783*	Closes an opened TIFF-F file.
BfvTiffOpen	*Vol 4: 785*	Opens a TIFF-F file.
BfvTiffReadIFD	*Vol 4: 787*	Reads the IFD of the current page in a TIFF-F file.
BfvTiffReadImage	*Vol 4: 790*	Reads image data of current page, one buffer at a time.
BfvTiffReadRes	*Vol 4: 792*	Interprets the IFD entry that contains the Y resolution tag; returns vertical resolution of current page.
BfvTiffWriteIFD	*Vol 4: 794*	Writes the IFD of the current page in a TIFF-F file.
BfvTiffWriteImage	*Vol 4: 797*	Writes image data of the current page to a TIFF-F file.
BfvTiffWriteRes	*Vol 4: 799*	Writes the page resolution specifications data to a TIFF-F file; returns the offset of the location where this data is written.
BfvToneDetectDisable	*Vol 3: 513*	Turns DTMF detection off.

Function	Location	Purpose
BfvToneDetectEnable	*Vol 3: 515*	Turns DTMF detection on.
BfvToneFlush	*Vol 3: 518*	Discards all tones currently stored in the buffer.
BfvToneGet	*Vol 3: 519*	Retrieves the next tone from the tone buffer and removes it from the buffer.
BfvTonePeek	*Vol 3: 521*	Retrieves the next tone from the tone buffer without disturbing the buffer.
BfvTonePlay	*Vol 3: 523*	Plays the tone for the specified time.
BfvTonePlayBeep	*Vol 3: 525*	Plays a single frequency tone.
BfvToneUnget	*Vol 3: 528*	Puts a tone at the top of the tone buffer, so it is available for the next request to retrieve a tone.
BsmiCloseAdapter	*Vol 5: 809*	Closes and releases an open handle.
BsmiControlRead	*Vol 5: 810*	Reads a control message from the module.
BsmiControlWrite	*Vol 5: 812*	Writes a control message to the module.
BsmiLineAlert	*Vol 5: 813*	Aborts a blocking ControlRead or ControlWrite function.
BsmiModuleList	*Vol 5: 815*	Returns a list of hardware modules in the current system.
BsmiOpenAdapter	*Vol 5: 817*	Returns a handle to the specific hardware module.
BsmiResetAdapter	*Vol 5: 819*	Resets the ISDN component of the module associated with a handle.
getopt	*Vol 1: 228*	Parses command line options.
resourceHandler	*Vol 3: 669*	Allows the Bfv API to ask the application to manage the resource allocation for resources required for a conference (user-defined function).

2 - Administration and Initialization

This chapter describes functions which allow you to attach, initialize and configure channels, and functions to query channel and system version information.

The line administration and initialization functions allow you to:

- Attach and detach from a line or a session.
- Configure a channel using a user-defined configuration file such as *btcall.cfg*.
- Reset the specified channel.
- Get information about the module and channel address for the specified channel.
- Get the number of available channels.

The line administration and initialization macros allow you to:

- Get information about the current version of the Bfv API or driver, and some information about the operating system environment.
- Get information about a line.

The BTLINE Structure

A separate BTLINE structure is created for a channel, and the Bfv API returns a pointer to the line structure when the application calls the *BfvLineAttach* or *BfvSessionAttach* function to open and attach a specified channel. All information about the channel is stored in its BTLINE structure, but only the line state, the line type, and channel number are actually relevant to the user. A BTLINE structure is deallocated by the *BfvLineDetach* or *BfvSessionDetach* function.

Applications do not directly access the internal fields of the BTLINE structure, but instead use the Bfv API functions. This also includes the following macros, described in detail in *Macros on page 82*:

LINE_HAS_CAP(*lp*, *cap*)
: Confirms whether or not the line has the specified capability *cap*.

LINE_STATE(*lp*)
: Returns the line state of the specified line.

LINE_TYPE(*lp*)
: Returns the line type of the specified line.

LINE_UNIT_NUM(*lp*)
: Returns the channel number of the specified line.

A line is always in one of the following states:

```
LINE_STATE_AWAIT_TRAINING
LINE_STATE_CONNECTED
LINE_STATE_FAX_MODE
LINE_STATE_HOLDUP
LINE_STATE_IDLE
LINE_STATE_NOLOOP
LINE_STATE_OFF_HOOK
LINE_STATE_RCV_INFO
LINE_STATE_RESETTING
LINE_STATE_RETAIN
LINE_STATE_RINGING
LINE_STATE_TURNAROUND
```

Each of the line states will be referred to by the descriptive part of its name only (for example, LINE_STATE_IDLE is referred to as IDLE).

The current state of the line is stored in the BTLINE structure. A pointer to this structure is passed as an argument to nearly all Bfv API entry points and is provided to the application by the *BfvLineAttach* or *BfvSessionAttach* function.

A number of functions serve as inputs to the BTLINE structure and affect the transition to different line states. Other functions check the current line state: some conditionally branch to other points in the code and some prevent inappropriate action (for example, frequent checking for CONNECTED before attempting to transmit data) from occurring.

The following provides a partial list of the functions and events and the line state they set:

FUNCTION	LINE STATE
BfvLineAnswer	Sets the state to CONNECTED.
BfvLineAttach	Initializes the state to IDLE.
BfvLineOriginateCall	Sets the state to CONNECTED or OFF_HOOK depending on the results from call progress monitoring.
BfvLineReset	Resets the state to IDLE.
BfvLineTerminateCall	Sets the state to IDLE.

EVENTS	LINE STATE
Answer tone detect	Sets the state to CONNECTED.
Direction change	Sets the state to TURNAROUND.
Disconnect	Sets the state to IDLE.
Received FSK data	Sets the state to AWAIT_TRAINING.
Ring detect	Sets the state to RINGING.
Training	Sets the state to FAX_MODE.

Function Summary

Table 1 provides a brief summary of the functions used for line administration and initialization.

Table 1. Line Administration and Initialization Function Summary

Function	Purpose	Page
BfvCheckAddress	Checks for the existence of an address.	*50*
BfvCheckFacility	Checks for the existence of the specified facility on the destination module and channel.	*52*
BfvLineAttach	Opens the given channel.	*54*
BfvLineConfig	Permits channel configuration via data buffers instead of a configuration file.	*57*
BfvLineDetach	Closes, hangs up, and resets the line.	*60*
BfvLineInfo	Returns the module type and channel address for a given channel from the driver.	*61*
BfvLineReset	Resets the channel and the state of the `BTLINE` structure.	*63*
BfvLinesAvail	Returns the number of enabled channels.	*67*
BfvModuleDeactivate	Deactivates a hardware module, marking it as dead.	*69*
BfvModuleInfo	Gets information about a module.	*71*
BfvSessionAttach	Creates a session for communicating with any facilities on a specified channel located on a specified module and machine.	*78*
BfvSessionDetach	Closes a specified channel and frees the `BTLINE` structure.	*81*

BfvCheckAddress

Purpose

Checks for the existence of the specified Millennium address.

Syntax

```
void
BfvCheckAddress                (lp, args)
   BTLINE                      *lp;
   struct args_addr_info       *args;
```

The structure contains the following fields.

Input Fields

```
MILL_ADDR m_addr;
```

Output Fields

```
unsigned exists;
RES res;
```

Input

lp

Pointer to the BTLINE structure. Can be NULL. Uses fewer resources if a valid *lp* is supplied.

args

Pointer to an argument structure containing input and output fields.

args.m_addr

Millennium address to check (see *page 51*).

Output

Return value: None.

args.exists

If nonzero, address exists.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Wild cards (M_ADDR_WILDCARD) can be used for any address components.

Example

```
struct args_addr_info args;

/* Check for the address representing the speech facility
   on module 2, channel 2, machine 1. */
BT_ZERO(args);
args.m_addr.mm_bFacility = MILL_FACILITY_SPEECH;
args.m_addr.mm_bModule = 2;
args.m_addr.mm_bChannel = 2;
args.m_addr.mm_bMachine = 1;

BfvCheckAddress(NULL,&args);
if (args.exists)
    printf("Address does exist.\n");
else
    printf("Address does not exist.\n");
```

BfvCheckFacility

Purpose

Checks for the existence of the specified facility on the destination module and channel.

Syntax

```
void
BfvCheckFacility            (lp, args)
   BTLINE                   *lp;
   struct args_addr_info    *args;
```

The structure contains the following fields.

Input Fields

```
unsigned facility;
```

Output Fields

```
unsigned exists;
RES res;
```

Input

lp

Pointer to the BTLINE structure of the channel to check.

args

Pointer to an argument structure containing input and output fields.

args.facility

The facility to check.

Output

Return value: None.

args.exists

If nonzero, the facility exists.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Example

```
BTLINE *lp;
struct args_addr_info args;

/* Check for the speech facility on the attached
   channel. */
BT_ZERO(args);
args.facility = MILL_FACILITY_SPEECH;
BfvCheckFacility(lp,&args);
if (args.exists)
    printf("Facility does exist.\n");
else
    printf("Facility does not exist.\n");
```

BfvLineAttach

Purpose

Opens the specified channel and initializes the BTLINE structure.

Syntax

```
BTLINE *
BfvLineAttach                (args)
   struct args_line_admin  *args;
```

The structure contains the following fields.

Input Fields

int *unit*;

Output Fields

RES *res*;

Modified Fields

unit, dest_addr, local_addr, present, unique, reset_on_close.

Input

args

Pointer to an argument structure containing input and output fields.

args.unit

The unit number range is 0...N –1, where N = the number of channels in the system. This value is also referred to as the ordinal channel number. If unit = –1, the next available line is attached.

Output

Return value:

A pointer to the BTLINE structure is returned if the attach is successful.

NULL is returned if the attach fails due to error. This condition can occur if the unit number is invalid, the line is already attached, or a system error occurs.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

When the function returns, check the LINE_TYPE macro to make sure the board is the type you want (BOARD_TYPE_BOSTON). BOARD_TYPE_TR114 is provided for compatibility.

A single process can open more than one channel. The *BfvLineReset* function normally follows *BfvLineAttach* to ensure the channel is properly initialized.

Note: Do not use line pointer sharing, which causes the same line pointer to be used by two threads at the same time. Attempting to do so will cause anomalous application behavior.

See Also

BfvLineReset, *BfvLineDetach*, *BfvLineConfig*, *BfvSessionAttach*

Example

```
BTLINE *lp;
int unit;
struct args_line_admin args;

BT_ZERO(args);
args.unit = unit;
if ((lp = BfvLineAttach(&args)) == NULL)
{
    fprintf(stderr, "Can't attach to channel\n");
}
```

BfvLineConfig

Purpose

Permits configuration of the channel in the same manner as the user-defined configuration file, but uses data buffers instead of a file.

Syntax

```
int
BfvLineConfig                (lp, args)
   BTLINE                    *lp;
   struct args_line_admin    *args;
```

The structure contains the following fields.

Input Fields

```
char *config_file_line;
int no_fonts;
int skip_teleph;
int no_init;
```

Output Fields

```
long reset_status;
RES res;
```

Input

lp

Pointer to the BTLINE structure of the channel to configure.

args

Pointer to an argument structure containing input and output fields.

args.config_file_line

A null-terminated ASCII string, containing a line of configuration information in the same format as the user-defined configuration file used with *BfvLineReset*.

This format is described in *Volume 6, Appendix A, Configuration Files.*

args.no_fonts

If set to 1, the Bfv API will not automatically attempt to download any fonts to the channel. The value must be the same for each call to this function and for the preceding call to *BfvLineReset*.

args.skip_teleph

If nonzero, indicates that the function should not attempt telephony/digital configuration. The value must be the same for each call to this function and for the preceding call to *BfvLineReset*.

args.no_init

If nonzero, instructs the function to not reset the channel and limit any other interaction with it. The value must be the same for each call to this function and for the preceding call to *BfvLineReset*.

Output

Return value:

0	Normal return.
–1	An error occurred. Check *args.reset_status* for more information.

args.reset_status

A value containing status information about this call. Bits in the value indicate particular conditions that have occurred. The value is valid only upon completion of the set of calls to *BfvLineConfig* (when called with NULL).

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

To use the *BfvLineConfig* function, call *BfvLineReset* with the *args.use_config_lines* input field set to 1. When *BfvLineReset* is called in this manner, *BfvLineConfig* must be called.

The application must call this function repeatedly with configuration lines, until the entire set of configuration information is processed. After all configuration lines are processed, this function must be called once with a NULL *args.config_file_line* value.

This function cannot be used with the *call_control* configuration option. To configure the telephony on a module, use the *BfvCallCtrlInit* function before calling this function and *BfvLineReset*.

See Also

BfvLineReset, *BfvCallCtrlInit*

Example

```
BTLINE *lp;
struct args_line_admin args;

BT_ZERO(args);
args.unit = 0;
lp = BfvLineAttach(&args);

args.config_file_name = "btcall.cfg";
args.use_config_lines = 1;
BfvLineReset(lp,&args);

args.config_file_line = "id_string my local id";
BfvLineConfig(lp,&args);

args.config_file_line = "font_file ibmpcps.fz8";
BfvLineConfig(lp,&args);
...
args.config_file_line = NULL;
BfvLineConfig(lp,&args);
```

BfvLineDetach

Purpose

Closes the specified channel and frees the BTLINE structure.

Syntax

```
void
BfvLineDetach                (lp, args)
   BTLINE                    *lp;
   struct args_line_admin    *args;
```

The structure contains the following fields.

Output Field

RES *res*;

Input

lp

Pointer to the BTLINE structure of the channel to detach.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

The *BfvLineDetach* function is exactly the same as the *BfvSessionDetach* function.

See Also

BfvLineAttach

Example

```
BTLINE *lp;
struct args_line_admin args;

BT_ZERO(args);
BfvLineDetach(lp, &args);
```

BfvLineInfo

Purpose

For a given channel, retrieves and returns the board type and I/O port address from the driver.

Syntax

```
int
BfvLineInfo                    (args)
   struct args_line_admin  *args;
```

The structure contains the following fields.

Input Fields

```
int unit;
```

Output Fields

```
int type;
unsigned port;
unsigned base;
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.unit

The ordinal channel number. Unit numbering starts at 0 and continues from board to board.

Output

Return value:

0 The type and port arguments were successfully updated.

nonzero An error occurred while attempting to query the driver. The driver is probably installed incorrectly.

args.type

Type of the specified channel. The possible values are:

BOARD_TYPE_UNKNOWN	0
No board or malfunctioning board.	
BOARD_TYPE_BOSTON	3
Brooktrout board.	
BOARD_TYPE_TR114	3
For compatibility only.	

args.port

The port address of the specified channel. Currently always 0.

args.base

The board base address for the specified channel. Currently always 0.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

This function checks whether a particular ordinal channel exists.

See Also

BfvLinesAvail

Example

```
struct args_line_admin args;

BT_ZERO(args);
args.unit = 0;
BfvLineInfo(&args);
printf("Channel 0 has type %d, port %x, base %x\n",
  args.type, args.port, args.base);
```

BfvLineReset

Purpose

Resets the channel and the state of the BTLINE structure.

Syntax

```
int
BfvLineReset                (lp, args)
   BTLINE                   lp;
   struct args_line_admin   *args;
```

The structure contains the following fields.

Input Fields

```
char *config_file_name;
int use_config_lines;
int mill_load_fonts;
int skip_teleph;
int no_init;
```

Output Fields

```
long reset_status;
RES res;
```

Modified Fields

config_file_line

Input

lp

Pointer to the BTLINE structure of the channel to reset.

args

Pointer to an argument structure containing input and output fields.

args.config_file_name

An ASCII string that contains the name of the user-defined configuration file for that line. The configuration files contain setup and telephony parameters and are constructed using the configuration file formats described in *Volume 6, Appendix A, Configuration Files*. If the value is NULL, no configuration file is used.

`args.use_config_lines`

If set to 1, indicates that configuration is performed using the *BfvLineConfig* function, either in addition to or instead of a supplied configuration file.

`args.mill_load_fonts`

Indicates that font downloads are to be performed according to the user configuration file information. This value is only valid for line pointers attached to logical channel 1 using *BfvSessionAttach*.

`args.skip_teleph`

If nonzero, indicates that telephony initialization should not be attempted.

`args.no_init`

If set to 1, instructs the function to not reset the channel nor initialize the channel by sending parameter settings to it. This value will only allow capability settings to be put into place so that future calls will have proper information. If set to 2, no interaction with the destination channel or address will be performed at all, not even to determine capabilities. This input field is only intended to be used when a channel already in use by one line pointer is about to be used by another line pointer obtained from a new *BfvLineAttach* or *BfvSessionAttach* call.

Output

Return value:

0	Channel successfully reset.
<0	Error condition, channel reset unsuccessful.
–1	Check *args.reset_status* for more information.

`args.reset_status`

A value containing status information about this call. Bits in the value indicate particular conditions that have occurred.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function performs the following:

- Resets the channel.
- Initializes both the state of the `BTLINE` structure and the driver data structures for the specified channel.
- Reloads the configuration parameters from the specified user-defined configuration file (often called *btcall.cfg*) and the country telephony configuration file (default *BT_CPARM.CFG*).
- Downloads the fonts specified by the *font_file* parameter in the user-defined configuration file if the *args.mill_load_fonts* option is specified.

BT_CPARM.CFG contains the country-specific parameters and is provided by Dialogic. The location of *BT_CPARM.CFG* must be specified in the user-defined configuration file (see the *bt_cparm* keyword in *Volume 6, Appendix A, User-defined Configuration File*) or *BT_CPARM.CFG* must reside in the current directory. For information about the parameters in *BT_CPARM.CFG*, see *Volume 6, Appendix G, Country-Specific Parameter Files*.

The channel is completely reset and re-initialized (includes going on-hook), and all previous channel information is cleared.

Call this function after the *BfvLineAttach* function, but before using the channel between fax and voice connections, or at any other time that channel re-initialization is appropriate.

Note: You should call this function at the end of every phone call, whether or not an error occurred.

When *args.mill_load_fonts* is nonzero, all fonts indicated in the user configuration file will be downloaded. If none are specified, then font 0 will be loaded using the default font name (“ibmpcps.fz8”). In addition, font number 255 serves as a default font and will be loaded regardless of whether any other font numbers are specified. This font is used if a font number referenced for ASCII conversion has not been loaded. It has the same default filename as font 0 if not specified. See the *BfvFaxDownloadFont* function in *Volume 4, Fax Processing*, for more details about font downloading. See the `LINE_FONT_DOWNLOADED` macro (*Volume 4*) to determine if a font was successfully downloaded.

Configuration can be performed with either an ASCII configuration file, individual configuration lines via the *BfvLineConfig* function, both, or none. These are controlled by the *args.config_file_name* and *args.use_config_lines* input fields.

➢ *The order of configuration steps is:*

1. Set all parameters to default values.
2. Process parameters specified by the configuration file.
3. Process parameters specified by individual configuration lines (*BfvLineConfig*).

When a parameter is processed, the new value specified overrides the old value.

This function's return value indicates only severe errors during reset. Other conditions, which might or might not be considered errors, can occur. These conditions include:

- Failure to open the font file specified in the user-defined configuration file.
- Failure to open the *BT_CPARM.CFG* file.
- Telephony configuration errors.

These and other conditions can be determined by examining the *args.reset_status* output field. If *BfvLineConfig* is used, the *args.reset_status* value supplied by that function should be examined as well. The bits in the value indicate particular conditions that have occurred and are defined by the RST_... values in the *mill_api.h* header files.

See Also

BfvLineAttach, *BfvLineDetach*, *BfvLineConfig*, LINE_CPU_TYPE, LINE_FIRM_DOWNLOADED, LINE_FONT_DOWNLOADED, LINE_HAS_CAP

Example

```
BTLINE *lp;
struct args_line_admin args;

BT_ZERO(args);
args.config_file_name = "btcall.cfg";
BfvLineReset(lp, &args);
```

BfvLinesAvail

Purpose

Checks the driver and returns a dynamically allocated array that contains the types of all the channels.

Syntax

```
char *
BfvLinesAvail                (args)
   struct args_line_admin  *args;
```

The structure contains the following fields.

Output Fields

```
unsigned base;
RES res;
```

Modified Fields

unit, type, port.

Input

args

Pointer to an argument structure containing input and output fields.

Output

Return value:

Pointer to a char * array containing one byte per channel. The byte for each channel contains one of the following values.

BOARD_TYPE_UNKNOWN	0
No board or malfunctioning board.	
BOARD_TYPE_BOSTON	3
Brooktrout board.	
BOARD_TYPE_TR114	3
For compatibility only.	

args.base

The board base address for the specified channel. Currently always 0.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function checks whether a particular ordinal channel exists.

The returned array is dynamically allocated. The application is responsible for freeing this memory when finished with it.

Enough memory is allocated for the maximum number of channels supported by the driver. In practice, the application needs to examine only as many channels as are in the system.

See Also

BfvLineInfo

Example

```
char *chan_info;
struct args_line_admin args;

BT_ZERO(args);
chan_info = BfvLinesAvail(&args);
printf("channel 0 has type %d.\n", chan_info[0]);
free(chan_info);
```

BfvModuleDeactivate

Purpose

Deactivates a hardware module, marking it as dead.

Syntax

```
void
BfvModuleDeactivate        (args)
   struct args_addr_info   *args;
```

The structure contains the following fields.

Input Fields

```
unsigned mod_num;
int deact_mode;
unsigned slot_num;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.mod_num

If nonzero, specifies the module number to deactivate. Only one of *args.mod_num* or *args.slot_num* can be specified.

args.deact_mode

Used with *args.mod_num*. If the value is DEACT_MODE_SINGLE (the default), only the specified module number is deactivated. If it is DEACT_MODE_ALL, all modules on the board containing the specified module number are deactivated. Valid values are:

DEACT_MODE_SINGLE	0
DEACT_MODE_ALL	1

args.slot_num

If nonzero, specifies the cPCI slot number of the board containing the module(s) to deactivate. Specify only one value for either *args.mod_num* or *args.slot_num*.

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 does not support this function.

After using this function to deactivate a module, no further communication with the module is possible until it is reactivated. Reactivation can be done using *BfvFirmwareDownload* or *BfvFirmwareDownloadData*.

Deactivation of a hardware module is often done so as to safely perform a manual hot swap of a board (for bus types that support hot swap, such as cPCI).

See Also

BfvFirmwareDownload, *BfvFirmwareDownloadData*

Example

```
struct args_addr_info args;

/* Deactivate module 2. */
BT_ZERO(args);
args.mod_num = 2;
BfvModuleDeactivate(&args);
```

BfvModuleInfo

Purpose

Retrieves information about a module.

Syntax

```
void
BfvModuleInfo                (lp, args)
   BTLINE                    *lp;
   struct args_addr_info     *args;
```

The structure contains the following fields.

Input Fields

```
unsigned mod_num;
```

Output Fields

```
unsigned exists;
unsigned num_channels;
unsigned hardware;
unsigned bus_type;
unsigned hw_type;
unsigned hw_id;
unsigned ordinal_start;
unsigned num_ordinals;
unsigned num_ordinals_init;
unsigned short pci_vend_id;
unsigned short pci_dev_id;
unsigned short pci_sub_vend_id;
unsigned short pci_sub_id;
unsigned char pci_interf_type;
unsigned char pci_runtime_start;
unsigned char pci_runtime_size;
unsigned char pci_mem_start;
unsigned char pci_mem_size;
unsigned char pci_modctrl_start;
unsigned char pci_modctrl_size;
unsigned char pci_io_start;
unsigned char pci_io_size;
unsigned char pci_irq;
unsigned char boot_ver_major;
unsigned char boot_ver_middle;
unsigned char boot_ver_minor;
unsigned char boot_ver_build;
unsigned short boot_ver_auto_num;
char boot_ver_date[20];
char boot_ver_comment[50];
unsigned hw_info;
RES res;
```

Input

lp

Pointer to the `BTLINE` structure. Can be `NULL`. Uses fewer resources if a valid *lp* is supplied.

args

Pointer to an argument structure containing input and output fields.

args.mod_num

Module number.

Output

Return value: None.

args.exists

If nonzero, indicates that the module exists.

args.num_channels

Indicates the number of channels on the module. If this value is 0, then the module is dead.

The value is the total number of channels supported by the module, including its administrative channel. The total is normally 1 more than the number of work channels, which are mapped into ordinal channel numbers.

For example, a hardware module with 48 work channels will be indicated as having 49 channels, and a hardware module with no work channels (no firmware downloaded) will be indicated as having 1 channel.

`args.hardware`

If nonzero, indicates the module is a hardware module.

`args.bus_type`

If this is a hardware module, indicates the bus type of the module. Values are:

`MILL_BUS_UNKNOWN`	0
`MILL_BUS_ISA`	1
`MILL_BUS_PCI`	2
`MILL_BUS_VIRT_MOD`	3

`args.hw_type`

If this is a hardware module, indicates a bus-type specific hardware type value (PCI device ID, ISA data mover).

`args.hw_id`

If this is a hardware module, indicates the unique board identifier value, if supported.

`args.ordinal_start`

If this is a hardware module, indicates the starting ordinal channel value mapped to this module.

`args.num_ordinals`

If this is a hardware module, indicates the number of ordinal channel values mapped to this module.

args.num_ordinals_init

The number of ordinals that the module was initially configured with in the driver. If this differs from the number of ordinals expected for the channels currently present, it may indicate that firmware reconfiguration took place.

args.pci_vend_id

For PCI boards, indicates the PCI Vendor ID from configuration space.

args.pci_dev_id

For PCI boards, indicates the PCI Device ID from configuration space.

args.pci_sub_vend_id

For PCI boards, indicates the PCI Subsystem Vendor ID from configuration space.

args.pci_sub_id

For PCI boards, indicates the PCI Subsystem ID from configuration space.

args.pci_interf_type

For PCI boards, indicates the PCI Interface type from configuration space.

args.pci_runtime_start

For PCI boards, indicates the start of runtime registers.

args.pci_runtime_size

For PCI boards, indicates the size of runtime registers.

args.pci_mem_start

For PCI boards, indicates the start of packet memory.

args.pci_mem_size

For PCI boards, indicates the size of packet memory.

args.pci_modctrl_start

For PCI boards, indicates the start of module ctrl registers.

args.pci_modctrl_size

For PCI boards, indicates the size of module ctrl registers.

`args.pci_io_start`

For PCI boards, indicates the start of IO area.

`args.pci_io_size`

For PCI boards, indicates the size of IO area.

`args.pci_irq`

For PCI boards, indicates the interrupt.

`args.boot_ver_major`

Indicates the major version number of the hardware module's boot ROM.

`args.boot_ver_middle`

Indicates the middle version number of the hardware module's boot ROM.

`args.boot_ver_minor`

Indicates the minor version number of the hardware module's boot ROM.

`args.boot_ver_build`

Indicates the hardware module's boot ROM build number.

`args.boot_ver_auto_num`

Indicates the "auto number" of the boot ROM firmware. This value is used internally, and refers to internal build procedures.

`args.boot_ver_date`

Indicates the hardware module's boot ROM date.

`args.boot_ver_comment`

Reports comments about the hardware module's boot ROM.

`args.hw_info`

Reports hardware information supplied by the firmware.

Value contains bits defined as follows:

Low density PCI mezzanine LP01H present	0x1
High density PCI mezzanine HP02e present	0x2

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Gets information about the module specified by *`args.mod_num`*. The output fields indicate whether the module exists, how many channels it supports (including the administrative channel), and whether it is a hardware module.

For SR140, the *`args.hardware`* will be set to nonzero, the same as a hardware module.

When the firmware is downloaded to a module for the first time, the assigned ordinal channel numbers start wherever the assignment left off on the previous module. As the system initializes the modules, this numbering process creates a continuous ordering of the channel assignments across all the modules in the system. On later downloads, each module's ordinals begin at the same location, regardless of any decrease in the channel count of a lower-numbered module. Therefore, if you decrease the channel count for a lower numbered module, the process creates gaps in the channel numbering assignments, possibly affecting your application. If you attempt to increase the channel count above any module's initial channel count, the system ignores the added channels.

For the following situations, restart the driver whenever you want to:

- Retain a continuous assignment of channel numbers after decreasing the channel count on any module.
- Increase the number of channels above a module's initial channel count.

Example

See the *modinfo.c* program located in the *bapp.src* directory.

BfvSessionAttach

Purpose

Creates a session for communicating with any facilities on a specified channel located on a specified module and machine.

Syntax

```
BTLINE *
BfvSessionAttach            (args)
   struct args_line_admin  *args;
```

The structure contains the following fields.

Input Fields

```
MILL_ADDR dest_addr;
MILL_ADDR local_addr;
int present;
int unique;
int reset_on_close;
int err_on_deact;
unsigned buf_size;
```

Output Fields

```
RES res;
MILL_ADDR local_addr;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.dest_addr

Destination Millennium address. Address component values are normally in the range 0x2 – (`M_ADDR_WILDCARD-1`), 1 for administrative purposes, `M_ADDR_WILDCARD` for wildcard. The machine component can be specified as 0 for current machine ID.

When specifying a standard destination address, the module component will be nonzero. In this case the facility component is not used.

The module component can also be 0 to indicate that an ordinal channel value will be specified in a list of all non-administrative channels on all hardware modules (for more information on these channel numbers see *BfvLineAttach*). The ordinal value is a

2-byte value with the least significant byte stored in the channel component and the most significant byte stored in the facility component.

The MILL_ADDR structure is documented in *Volume 6, Appendix B, Bfv API Structures*.

`args.local_addr`

Requested local Millennium address. Normally set to 0 for driver assignment.

The MILL_ADDR structure is documented in *Volume 6, Appendix B, Bfv API Structures*.

`args.present`

If nonzero, indicates that the destination address must exist for this attach to succeed.

`args.unique`

If nonzero, indicates that the destination address must not already be attached for this attach to succeed.

`args.reset_on_close`

If nonzero, indicates that the destination channel, if hardware and non-administrative, will automatically be reset and its output buffers cleared when the session is detached. You should set this option unless multiple line pointers are being used to attach to the same channel.

`args.err_on_deact`

If nonzero, the application session will be notified if the attached address is deactivated. This condition can happen due to an explicit deactivation, if the destination does not behave properly, or of its own volition. Set this value to ensure that the application will terminate, rather than continue to wait for a response that will never arrive.

You should set this option whenever attaching to a hardware channel.

`args.buf_size`

Specifies a driver buffer size to be used when creating the application session. This will be in effect only if greater than the default driver buffer size. This feature should be used only in circumstances where extremely high traffic volume warrants.

Output

Return value:

A pointer to the `BTLINE` structure or `NULL` is returned on error.

`args.local_addr`

The local assigned Millennium address.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

To provide multiple access to the same channel, attach a second line pointer with *args.unique* set to 0 and *args.reset_on_close* possibly set to 0 as well. When *BfvLineReset* is called, often the *args.no_init* option will be used.

This function is useful for applications using full duplex speech but only when it is certain that use of the multiple line pointers will not interfere.

Note: Do not use line pointer sharing, which causes the same line pointer to be used by two threads at the same time. Attempting to do so will cause anomalous application behavior.

The "channel" parameter in the user-defined configuration file cannot be used with a line pointer from this function, unless it is attached to an ordinal channel.

See Also

BfvLineAttach

Example

```
BTLINE *lp;
struct args_line_admin args;

/* Attempt to attach to the administrative facility on
   module 2, channel 1, using the current machine. If it
   does not exist, the attach will fail. */
BT_ZERO(args);
args.dest_addr.mm_bFacility =
    MILL_FACILITY_ADMINISTRATION;
args.dest_addr.mm_bModule = 2;
args.dest_addr.mm_bChannel = 1;
args.dest_addr.mm_bMachine = 0;
args.present = 1;
if ((lp = BfvSessionAttach(&args)) == NULL
    printf("SessionAttach failed.\n");
```

BfvSessionDetach

Purpose

Closes the specified channel and frees the BTLINE structure.

Syntax

```
void
BfvSessionDetach            (lp, args)
   BTLINE                   *lp;
   struct args_line_admin   *args;
```

The structure contains the following fields.

Output Field

RES *res*;

Input

lp

Pointer to the BTLINE structure of the channel to detach.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

The *BfvSessionDetach* function is exactly the same as the *BfvLineDetach* function.

See Also

BfvSessionAttach

Example

```
BTLINE *lp;
struct args_line_admin args;

BT_ZERO(args);
BfvSessionDetach(lp, &args);
```

Macros

BT_ZERO (*item*)

This macro is equivalent to calling `memset` to clear (set contents to 0) an item of any type. This macro, or equivalent behavior, must be performed on argument structures before calling Bfv API functions (see *Using Bfv API Function Argument Structures on page 31*).

LINE_STATE (*lp*)

Accesses the current state of the line specified by *lp*.

This macro evaluates to an integer. Applications can compare this value to the `enum` values defined in the *mill_api.h* header file to determine the current line state or set the macro to one of the `enum` values. Valid values are:

```
LINE_STATE_AWAIT_TRAINING
LINE_STATE_CONNECTED
LINE_STATE_FAX_MODE
LINE_STATE_HOLDUP
LINE_STATE_IDLE
LINE_STATE_NOLOOP
LINE_STATE_OFF_HOOK
LINE_STATE_RCV_INFO
LINE_STATE_RESETTING
LINE_STATE_RETAIN
LINE_STATE_RINGING
LINE_STATE_TURNAROUND
```

When set manually, the `LINE_STATE` is most often set to `LINE_STATE_CONNECTED`.

LINE_TYPE *(lp)*

Returns a value that identifies the type of channel specified by *lp*. Valid return values are:

`BOARD_TYPE_UNKNOWN` 0

No board or malfunctioning board

`BOARD_TYPE_BOSTON` 3

Brooktrout board

`BOARD_TYPE_TR114` 3

For compatibility only

LINE_UNIT_NUM (*lp*)

Returns the channel number of the channel specified by *lp*.

If no ordinal channel number was specified when attaching, then the value returned will be –1.

LINE_CPU_TYPE (*lp*)

For compatibility only.

This macro returns 1.

LINE_HAS_CAP (*lp, cap*)

Determines if a channel has a particular capability. Returns nonzero if *lp* has capability *cap*; returns 0 if *lp* does not have capability *cap*. The *caps.h* header file contains definitions of the capabilities. You should use this macro instead of examining the `LINE_TYPE` value.

MOD_BOARD_SLOT (*mod_num*)

For cPCI boards only, returns the cPCI slot number corresponding to the given module number.

MOD_CPU_NUM (*mod_num*)

For cPCI boards only, returns the CPU number within the board corresponding to the given module number.

LP_BOARD_SLOT *(lp)*

For cPCI boards only, returns the cPCI slot number corresponding to the given line pointer.

LP_CPU_NUM *(lp)*

For cPCI boards only, returns the CPU number within the board corresponding to the given line pointer.

LINE_ALERT_CTL *(lp)*

Accesses a value that controls the application's behavior when a channel receives an alert.

If the value is 0 (default), the application behaves as described under *BfvLineAlert*. Most Bfv API functions return an error indication as soon as possible, and in this case, the application should call *BfvLineReset*.

The application can set the bits within this value to alter this default behavior. The following bit is defined:

b0 — Causes the speech record and playback functions to attempt to quit with no loss of data and with continued use of the channel without a reset after the function returns.

LINE_PRIVATE_USER_DATA *(lp)*

Accesses a private user data pointer specific to the provided line structure pointer which can be read or written at any time. This value is not cleared by *BfvLineReset*.

BT_ARGS *((arg-list))*

BT_CBARGS *((arg-list))*

Applications use the `ARGS` macro for the arguments to a function and the `CBARGS` (callback args) macro for a function pointer argument to a function. The Bfv API supplies two macros since different compilation environments might handle these conditions differently.

Both macros expand to (*arg-list*) if the compilation environment supports function prototypes; otherwise, they expand to ().

Therefore, a function *foo*, which takes two arguments, *bar1* and *bar2*, could have a portable prototype of:

```
int foo ARGS((int bar1, int bar2));
```

Applications use the following macros to deal with architecture-dependent, multibyte integer storage formats.

BT_LITTLE_ENDIAN

Defined only when the application is compiled on a system with little endian integers (low byte first). Almost all PCs use this format.

BT_BIG_ENDIAN

Defined only when the application is compiled on a system with big endian integers (high byte first).

BYTE_SWAP_SHORT *(v)*

This macro reverses the byte order of a short (two-byte) integer variable. Defined only if `BT_BIG_ENDIAN` is defined.

BYTE_SWAP_LONG *(v)*

This macro reverses the byte order of a long (four-byte) integer variable. Defined only if `BT_BIG_ENDIAN` is defined.

USES_FAT_FILESYSTEM

Defined only when the application is compiled on a system that uses the Microsoft FAT or NTFS file system such as Windows.

BT_API_SET_VER ()

When called once at the start of a Bfv API application, this macro informs the Bfv API about the version being used to compile the program. Use this macro to allow the application to support binary compatibility, enabling an application written and compiled with SDK 6.0 or later to work with a later SDK version without a recompile or relink.

For more information on binary compatibility, refer to the *Compatibility for Compiling* section in the *Dialogic® Brooktrout® Fax Products SDK Developer's Guide.*

BROOKTROUT_MILLENNIAL

Indicates the Bfv API environment. This symbol defines that the source is being compiled in a Millennial (Dialogic® Brooktrout® boards) environment.

The following macros are rarely used. They provide direct access to configuration structures used internally within the Bfv API.

LINE_PHONE_STRUCT *(lp)*

Returns a pointer to a structure of type *country_phone_info* containing country specific telephony parameters. See the *phone.h* header file for the structure definition.

LINE_CONFIG_STRUCT *(lp)*

Returns a pointer to a structure of type *user_config_params* containing user supplied configuration parameters. See the *line.h* header file for the structure definition.

The following macros return the Bfv API or driver version number either as a character string or as decimal number. The Bfv API and driver version numbers should be the same.

API_VER_NUM

Returns the Bfv API version number as a decimal number, for example, 400000900.

API_VERSION

Returns the Bfv API version number as a character string, for example, “4.9.00”.

MILL_API_BUILD_NUM

Returns the build number of the Bfv API.

API_V1

Returns the high, or major, portion of the Bfv API version number. This value refers to the driver version installed when the application is compiled. For example, if the version number is 4.9.00, this macro returns 4.

API_V2

Returns the second highest, or minor, portion of the Bfv API version number. This value refers to the driver version installed when the application is compiled. For example, if the version number is 4.9.00, this macro returns 9.

API_V3

Returns the third highest portion of the Bfv API version number, which is the upper digit of the revision number. This value refers to the driver version installed when the application is compiled. For example, if the version number is 4.9.00, this macro returns 0.

API_V4

Returns the fourth highest portion of the Bfv API version number, which is the lower digit of the revision number. This value refers to the driver version installed when the application is compiled. For example, if the version number is 4.9.00, this macro returns 0.

LINE_API_VER_LOADED ()

Allows an application to retrieve information about the linked/loaded Bfv API library version (see *API_VER_NUM*). The value returned is the library version number plus 100000 times the library build/interface number. For example, if the library version is 4.9 build 1, this macro returns 400100900.

MILL_VER_NUM

Returns the driver version number as a decimal number, for example, 4900.

This value refers to the driver version installed when the application is compiled. See also *LINE_DRIVER_VER_LOADED* **(lp)**.

MILL_DRIVER_VERSION

Returns the driver version number as a character string, for example, “4.9.00”.

This value refers to the driver version installed when the application is compiled. See also *LINE_DRIVER_VER_LOADED* **(lp)**.

MILL_BUILD_NUM

Returns the build number of the driver.

This value refers to the driver version installed when the application is compiled. See also *LINE_DRIVER_VER_LOADED* **(lp)**.

MILL_V1

Returns the high, or major, portion of the driver version number. For example, if the version number is 4.9.00, this macro returns 4.

This value refers to the driver version installed when the application is compiled. See also *LINE_DRIVER_VER_LOADED* **(lp)**.

MILL_V2

Returns the second highest, or minor, portion of the driver version number. For example, if the version number is 4.9.00, this macro returns 9.

This value refers to the driver version installed when the application is compiled. See also *LINE_DRIVER_VER_LOADED* **(lp)**.

MILL_V3

Returns the third highest portion of the driver version number, which is the upper digit of the revision number. For example, if the version number is 4.9.00, this macro returns 0.

This value refers to the driver version installed when the application is compiled. See also *LINE_DRIVER_VER_LOADED* **(lp)**.

MILL_V4

Returns the fourth highest portion of the driver version number, which is the lower digit of the revision number. For example, if the version number is 4.9.00, this macro returns 0.

This value refers to the driver version installed when the application is compiled. See also *LINE_DRIVER_VER_LOADED* **(lp)**.

LINE_DRIVER_VER_LOADED *(lp)*

Allows an application to retrieve information about the loaded driver version (see *MILL_VER_NUM*). The value returned is the driver version number multiplied by 10000 plus the driver build number. For example, if the driver version is 4.9 build 1, this macro returns 49000001.

Low-Level Macros

The following low-level macros are rarely used. If you use them, do so with care.

LINE_DEST_ADDR *(lp)*

Returns the attached destination Millennium address, type `MILL_ADDR`. This value is either the address implicitly attached to via *BfvLineAttach* or the address explicitly attached to via *BfvSessionAttach* (see *Volume 6, Appendix B, Bfv API Structures*).

LINE_APP_ADDR *(lp)*

Returns the local application session Millennium address, type `MILL_ADDR`. This value is the address assigned to the application session by the driver when *BfvLineAttach* or *BfvSessionAttach* is done (see *Volume 6, Appendix B, Bfv API Structures*).

LINE_SRC_ADDR *(lp)*

Accesses, for setting or reading, the default source address used for sending packets from this application. This value is normally the same value as that returned by *LINE_APP_ADDR* **(lp)**, but in rare circumstances an application can choose to set this address to a different value.

LINE_SET_INCOMING_CMD_FUNC *(lp, func)*

Sets up a user-supplied function that the Bfv API calls for every incoming command it processes for the current application session. The Bfv API calls this function after performing its own internal processing of the command.

If *func* is set to `NULL`, this feature is disabled. This setting is not cleared by *BfvLineReset*.

The user supplied function is called as:

```
void (*func)(BTLINE *lp,struct args_packet *args);
```

The *lp* argument contains the line pointer, and the *args* argument is a pointer to an *args_packet* structure containing a parsed command. The structure and its contents must not be modified except as described in *Handling Alerts* (see *page 92*).

Use this macro to check for particular incoming commands or status changes while a Bfv API function call is in progress.

LINE_INCOMING_CMD_FUNC *(lp)*

Similar to *Handling Alerts*. Allows access to the incoming command function for setting or reading.

Handling Alerts

When an alert is received, the user-supplied function will be called with a particular command described by *args*. It will contain *args->facility* == `MILL_FACILITY_HOST_CTRL`, *args->cmd_verb* == `MILL_VERB_EVENT`, and *args->cmd_specifier* == `HOST_CTRL_ALERT_EVENT`. The *args->var_value* field will contain the alert value supplied when the alert was triggered.

If the function takes no action, alert processing will proceed and the normal outcome of an alert, stopping current activities, will take place as described under *BfvLineAlert*. If the function sets *args->facility* to 0, the alert will be cancelled, and no further alert processing will occur.

The alert can be modified to appear to be a different incoming command. In this case, the Bfv API will process the command, and the incoming command function will be called again with the new values, just as if the command had actually been received. The modification can be done either by modifying *args->facility*, *args->cmd_verb*, *args->cmd_specifier*, *args->bit_len*, *args->tag_ptr*, and *args->total_tag_len*, or by modifying *args->facility*, *args->cmd_buf*, and *args->cmd_len*. The former method is for an already parsed command (except possibly for tag values), and the latter is for a command that is unparsed.

3 - Firmware

This chapter describes the functions used to download firmware and feature set data, and to get feature data or firmware configuration options.

With the specialized firmware functions, you can download firmware to the module from a file or a buffer and get information about a module's firmware configuration options.

Firmware macros can provide the version number, build number and date of the following:

- Control processor firmware
- Boot ROM firmware
- DSP (version and number of DSPs on the module)

Function Summary

Table 2 provides a brief summary of the functions used to download firmware and feature set data, get feature data, and get the firmware's configuration options.

Table 2. Firmware Function Summary

Function	Purpose	Page
BfvFeatureSetDownload	Downloads feature set data supplied in a specified file.	*95*
BfvFeatureSetDownloadData	Downloads feature set data supplied in a specified user buffer.	*97*
BfvFeatureSetQuery	Queries the feature data currently stored in the module's feature set hardware.	*99*
BfvFirmwareDownload	Downloads firmware to a module from a file.	*101*
BfvFirmwareDownloadData	Downloads firmware to a module from a buffer.	*105*
BfvModuleConfigSpecsGet	Gets information about a module's possible firmware configuration options, as reported by the firmware.	*109*

BfvFeatureSetDownload

Purpose

Downloads feature set data supplied in a specified file.

Syntax

```
int
BfvFeatureSetDownload      (lp, args)
   BTLINE                  *lp;
   struct args_feature_set *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fname

Name of the file containing feature data to download.

Output

Return value:

0 Success.

>0 Operational error reported by firmware.

ADMIN_FTRL_NO_MEMORY_DEF	0x01
ADMIN_FTRL_READ_SSEEPROM_ERR_DEF	0x02
ADMIN_FTRL_WRITE_SSEEPROM_ERR_DEF	0x03
ADMIN_FTRL_SSEEPROM_WRITTEN_DEF	0x04
ADMIN_FTRL_MAGIC_NUM_ERR_DEF	0x05
ADMIN_FTRL_CHKSUM_ERR_DEF	0x06

<0 Other error.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

SR140 does not support this function.

This function operates on raw binary feature set data files. Dialogic commonly supplies license files in ASCII format. These files contain an ASCII representation of the feature set data, but this function will not operate directly on the ASCII file format.

Feature set data contains licensing information specific to a given board. The licensing information contains information about features the user can access, the number of channels available, and other pertinent information.

See Also

BfvFeatureSetDownloadData, *BfvFeatureSetQuery*

Example

See the *feature.c* program located in the *bapp.src* directory.

BfvFeatureSetDownloadData

Purpose

Downloads feature set data supplied in a specified user buffer.

Syntax

```
int
BfvFeatureSetDownloadData (lp, args)
   BTLINE                  *lp;
   struct args_feature_set *args;
```

The structure contains the following fields.

Input Fields

```
unsigned char *buf;
unsigned size;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.buf

Pointer to the data buffer containing feature data to download.

args.size

Size of the data buffer in bytes.

Output

Return value:

0 Success.

>0 Operational error reported by firmware.

ADMIN_FTRL_NO_MEMORY_DEF	0x01
ADMIN_FTRL_READ_SSEEPROM_ERR_DEF	0x02
ADMIN_FTRL_WRITE_SSEEPROM_ERR_DEF	0x03
ADMIN_FTRL_SSEEPROM_WRITTEN_DEF	0x04
ADMIN_FTRL_MAGIC_NUM_ERR_DEF	0x05
ADMIN_FTRL_CHKSUM_ERR_DEF	0x06

<0 Other error.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 does not support this function.

This function operates on raw binary feature set data files. Dialogic commonly supplies license files in ASCII format. These files contain an ASCII representation of the feature set data, but this function will not operate directly on the ASCII file format.

Feature set data contains licensing information specific to a given board. The licensing information contains information about features the user can access, the number of channels available, and other pertinent information.

The application calls the function repeatedly until all data is supplied, after which the function must be called again with *args.buf* set to NULL.

See Also

BfvFeatureSetDownload, *BfvFeatureSetQuery*

Example

See the *feature.c* program located in the *bapp.src* directory.

BfvFeatureSetQuery

Purpose

Queries the feature data currently stored in the module's feature set hardware.

Syntax

```
void
BfvFeatureSetQuery          (lp, args)
   BTLINE                   *lp;
   struct args_feature_set  *args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

```
char key[9];
int type;
int value_int;
char output_value_string[128];
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.key

ASCII feature name, 0-terminated, up to eight characters long.

args.type

Type of the feature, one of:

FEATURE_BOOLEAN

FEATURE_INTEGER

FEATURE_STRING

`args.value_int`

The value of the feature if it is of type `FEATURE_BOOLEAN` or `FEATURE_INTEGER`.

`args.output_value_string`

The value of the feature if it is of type `FEATURE_STRING`.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The application calls the function repeatedly, retrieving one feature entry with each call, until finally *args.key* is an empty string.

The value for each feature on an SR140 module reflects the total value of this feature for all active SR140 licenses, not for the module itself. It may be higher than a single module can utilize.

See Also

BfvFeatureSetDownload, *BfvFeatureSetDownloadData*

Example

See the *feature.c* program located in the *bapp.src* directory.

BfvFirmwareDownload

Purpose

Downloads firmware to a module.

Syntax

```
int
BfvFirmwareDownload          (lp, args)
   BTLINE                    *lp;
   struct args_download      *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
int specify_type;
unsigned dl_type;
unsigned dl_dest;
unsigned config_spec_value;
```

Output Fields

```
RES res;
```

Modified Fields

buf, *size*.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fname

Name of the firmware file to download.

args.specify_type

If nonzero, *args.dl_type* and *args.dl_dest* will be used.

`args.dl_type`

Download type:

`ADMIN_DSP_LOADER_DEF`	0
DSP loader code	
`ADMIN_DSP_APP_DEF`	1
DSP application code	
`ADMIN_PROC_APP_DEF`	2
Main processor application code	
`ADMIN_PROC_FLASH_DEF`	3
Main processor flash code (not used on current Brooktrout modules)	

`args.dl_dest`

Indicates the destination DSPs, each bit corresponding to a DSP unit (usually a single DSP). The value 0xFFFFFFFF is recommended.

`args.config_spec_value`

Selects a firmware configuration from among several possible alternatives. Possible values can be obtained using *BfvModuleConfigSpecsGet*. The value 0 represents the firmware's default configuration. For use only with *args.dl_type* `ADMIN_PROC_APP_DEF`. The current meaning of the configuration specification value specifies the number of channels for firmware configuration.

Output

Return value:

0	Success.
>0	Operational error reported by firmware.

`ADMIN_FW_DOWNLOAD_BAD_TYPE_DEF`	0x01
`ADMIN_FW_DOWNLOAD_HEADER_ERR_DEF`	0x02
`ADMIN_FW_DOWNLOAD_WRONG_VERSION_DEF`	0x03
`ADMIN_FW_DOWNLOAD_CODE_POINTER_ERR_DEF`	0x04
`ADMIN_FW_DOWNLOAD_CRC_ERR_DEF`	0x05
`ADMIN_FW_DOWNLOAD_FORMAT_ERR_DEF`	0x06
`ADMIN_FW_DOWNLOAD_INCOMPATIBLE_BSP_DEF`	0x07

ADMIN_FW_DOWNLOAD_NO_FLASH_IMAGE_DEF	0x08
ADMIN_FW_DOWNLOAD_BAD_DIMENSION_DEF	0x09
ADMIN_FW_DOWNLOAD_FLASH_UPD_ERR_DEF	0x0A
ADMIN_FW_DOWNLOAD_REBOOT_REQUIRED_DEF	0x0C
ADMIN_FW_DOWNLOAD_POWER_CYCLE_REQUIRED_DEF	0x0D

<0 Other error.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 does not support this function.

The line structure must have been created via *BfvSessionAttach* with *args.dest_addr* set to indicate a hardware module with channel 1. A filename is specified using *args.fname*. If *args.specify_type* is nonzero, then *args.dl_type* and *args.dl_dest* indicate the type of firmware being downloaded and the download locations.

If the download type is PROC_APP, the driver attempts to re-establish communications with the destination module. If the module was previously marked as dead, it might become usable again.

The firmware consists of several types, by number, which must be downloaded in the proper sequence. The firmload utility (see *Sample Applications and Utilities* in the *Dialogic® Brooktrout® Fax Products SDK Developer Guide*) automatically identifies the proper files and downloads them in the correct sequence.

The sequence is as follows:

Type 2 (PROC_APP)	=	Control processor firmware.
Type 0 (DSP_LOADER)	=	DSP boot loader (not used for version 1.4.0 and later).
Type 1 (DSP_APP)	=	DSP firmware.

The download type `PROC_APP` can be used for flash boot ROM updates. When this, updating is done under some conditions, one of the following values might be returned that require special handling:

- `ADMIN_FW_DOWNLOAD_REBOOT_REQUIRED_DEF` indicates that a system reboot is required.
- `ADMIN_FW_DOWNLOAD_POWER_CYCLE_REQUIRED_DEF` indicates that a system power cycle is required.

See Also

BfvFirmwareDownloadData, *BfvModuleConfigSpecsGet*

Example

See the *firm.c* program located in the *bapp.src* directory.

BfvFirmwareDownloadData

Purpose

Downloads firmware to a module.

Syntax

```
int
BfvFirmwareDownloadData    (lp, args)
   BTLINE                  *lp;
   struct args_download    *args;
```

The structure contains the following fields.

Input Fields

```
unsigned char *buf;
unsigned size;
int specify_type;
unsigned dl_type;
unsigned dl_dest;
unsigned config_spec_value;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fname

Name of the firmware file to download.

args.specify_type

If nonzero, *args.dl_type* and *args.dl_dest* will be used.

`args.dl_type`

Download type:

`ADMIN_DSP_LOADER_DEF`	0
DSP loader code	
`ADMIN_DSP_APP_DEF`	1
DSP application code	
`ADMIN_PROC_APP_DEF`	2
Main processor application code	
`ADMIN_PROC_FLASH_DEF`	3
Main processor flash code (not used on current Brooktrout modules)	

`args.dl_dest`

Indicates the destination DSPs, each bit corresponding to a DSP unit (usually a single DSP). The value 0xFFFFFFFF is recommended.

`args.config_spec_value`

Selects a firmware configuration from among several possible alternatives. Possible values can be obtained using *BfvModuleConfigSpecsGet*. The value 0 represents the firmware's default configuration. For use only with *args.dl_type* `ADMIN_PROC_APP_DEF`. The current meaning of the configuration specification value specifies the number of channels for firmware configuration.

Output

Return value:

0	Success.
>0	Operational error reported by firmware.

`ADMIN_FW_DOWNLOAD_BAD_TYPE_DEF`	0x01
`ADMIN_FW_DOWNLOAD_HEADER_ERR_DEF`	0x02
`ADMIN_FW_DOWNLOAD_WRONG_VERSION_DEF`	0x03
`ADMIN_FW_DOWNLOAD_CODE_POINTER_ ERR_DEF`	0x04
`ADMIN_FW_DOWNLOAD_CRC_ERR_DEF`	0x05
`ADMIN_FW_DOWNLOAD_FORMAT_ERR_DEF`	0x06
`ADMIN_FW_DOWNLOAD_INCOMPATIBLE_BSP_DEF`	0x07

ADMIN_FW_DOWNLOAD_NO_FLASH_IMAGE_DEF	0x08
ADMIN_FW_DOWNLOAD_BAD_DIMENSION_DEF	0x09
ADMIN_FW_DOWNLOAD_FLASH_UPD_ERR_DEF	0x0A
ADMIN_FW_DOWNLOAD_REBOOT_REQUIRED_DEF	0x0C
ADMIN_FW_DOWNLOAD_POWER_CYCLE_ REQUIRED_DEF	0x0D

<0 Other error.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 does not support this function.

The line structure must have been created via *BfvSessionAttach* with *args.dest_addr* set to indicate a hardware module with channel 1. Data is supplied via successive calls with *args.buf* and *args.size* set to appropriate values. To finish, call this function with *args.buf* set to NULL and *args.size* set to 0. If *args.specify_type* is nonzero, then *args.dl_type* and *args.dl_dest* indicate the type of firmware being downloaded and the download location.

If the download type is PROC_APP, the driver attempts to re-establish communications with the destination module. If the module was previously marked as dead, it might become usable again.

The firmware consists of several types, by number, which must be downloaded in the proper sequence. The firmload utility (see *Sample Applications and Utilities* in the *Dialogic® Brooktrout® Fax Products SDK Developer Guide*) automatically identifies the proper files and downloads them in the correct sequence.

The sequence is as follows:

Type 2 (PROC_APP)	=	Control processor firmware.
Type 0 (DSP_LOADER)	=	DSP boot loader (not used for version 1.4.0 and later).
Type 1 (DSP_APP)	=	DSP firmware.

The download type PROC_APP can be used for flash boot ROM updates. When this, updating is done under some conditions, one of the following values might be returned that require special handling:

- ADMIN_FW_DOWNLOAD_REBOOT_REQUIRED_DEF indicates that a system reboot is required.
- ADMIN_FW_DOWNLOAD_POWER_CYCLE_REQUIRED_DEF indicates that a system power cycle is required.

See Also

BfvFirmwareDownload, *BfvModuleConfigSpecsGet*

Example

```
BTLINE *lp;
int n;
char *buf;
struct args_download args;

/* Put DSP firmware data into a buffer and download it. */
while ((n = get_data(buf)) > 0)
{
    BT_ZERO(args);
    args.buf = buf;
    args.size = n;
    args.specify_type = 1;
    args.dl_type = ADMIN_DSP_APP_DEF;
    args.dl_dest = 0xffffffff;
    BfvFirmwareDownloadData(lp,&args);
}
```

BfvModuleConfigSpecsGet

Purpose

Retrieves information about a module's possible firmware configuration options, as reported by the firmware.

Syntax

```
int
BfvModuleConfigSpecsGet     (lp, args)
   BTLINE                   *lp;
   struct args_addr_info    *args;
```

The structure contains the following fields.

Input Fields

```
unsigned mod_num;
unsigned config_spec_num;
```

Output Fields

```
unsigned config_spec_value;
char config_spec_desc [51];
RES res;
```

Input

lp

Pointer to the BTLINE structure. Can be NULL. Uses fewer resources if a valid *lp* is supplied.

args

Pointer to an argument structure containing input and output fields.

args.mod_num

Module number.

args.config_spec_num

Index of the configuration specification to be retrieved, starting from 0.

Output

Return value:

0	Configuration returned.
1	No more configurations.
<0	Error.

args.config_spec_value

Configuration specification value.

args.config_spec_desc

Optional configuration specification string.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

To get information about the next configuration, specify a configuration specification index number, starting from 0, in *args.config_spec_num*. The output value *args.config_spec_value* will contain the next configuration value and *args.config_spec_desc* will contain an optional string description of the configuration.

Call the function repeatedly with increasing values of *args.config_spec_num*.

The function will return 0 if a configuration is returned, and will return 1 when there are no more configurations.

The configuration specification values can be supplied to *BfvFirmwareDownload* or *BfvFirmwareDownloadData* when downloading type 2 (PROC_APP) firmware.

See Also

BfvFirmwareDownload, *BfvFirmwareDownloadData*

Example

See the *modinfo.c* program located in the *bapp.src* directory.

Macros

LINE_FIRM_BOOT_ROM_MAJOR *(lp)*

Returns the major portion of the Boot ROM firmware version number. For example, if the version number is 6.3 or 6.4.3, this macro returns 6.

LINE_FIRM_BOOT_ROM_MIDDLE *(lp)*

Returns the middle portion of the Boot ROM firmware version number. For example, if the version number is 6.4.3, this macro returns 4.

LINE_FIRM_BOOT_ROM_MINOR *(lp)*

Returns the minor portion of the Boot ROM firmware version number. For example, if the version number is 6.3 or 6.4.3, this macro returns 3.

LINE_FIRM_BOOT_ROM_BUILD *(lp)*

Returns the build number of the Boot ROM firmware version.

LINE_FIRM_BOOT_ROM_AUTO_NUM *(lp)*

Returns the boot ROM firmware “auto number”. This value is used internally and refers to internal build procedures.

LINE_FIRM_BOOT_ROM_DATE *(lp)*

Returns the date of the Boot ROM firmware version. The date is a character string in the form M/D/Y (for example 1/31/02).

LINE_FIRM_BOOT_ROM_COMMENT *(lp)*

Returns the comment of the Boot ROM firmware version.

Note: The `LINE_FIRM_CTRL_PROC_*` macros that follow are only available when control processor firmware has been downloaded to the module, the line pointer has been attached to a channel other than channel 1, and when *BfvLineReset* is used without the *no_init* option. In all other cases, only the `LINE_FIRM_BOOT_ROM_*` macros will be available.

LINE_FIRM_CTRL_PROC_MAJOR *(lp)*

Returns the major portion of the control processor firmware version number. This macro is the same as `LINE_FIRM_MAJOR`.

LINE_FIRM_CTRL_PROC_MIDDLE *(lp)*

Returns the middle portion of the control processor firmware version number. This macro is the same as `LINE_FIRM_MIDDLE`.

LINE_FIRM_CTRL_PROC_MINOR *(lp)*

Returns the minor portion of the control processor firmware version number. This macro is the same as `LINE_FIRM_MINOR`.

LINE_FIRM_CTRL_PROC_BUILD *(lp)*

Returns the build number of the control processor firmware version.

LINE_FIRM_CTRL_PROC_AUTO_NUM *(lp)*

Returns the control processor firmware “auto number”. This value is used internally and refers to internal build procedures.

LINE_FIRM_CTRL_PROC_DATE *(lp)*

Returns the date of the control processor firmware version. The date is a character string in the form M/D/Y (for example, 1/31/02). This macro is the same as `LINE_FIRM_DATE`.

LINE_FIRM_CTRL_PROC_COMMENT *(lp)*

Returns comments about the control processor firmware version.

Note: The `LINE_FIRM_DSP_*` macros that follow are only available when DSP firmware has been downloaded to the module, the line pointer has been attached to a channel other than channel 1, and when *BfvLineReset* is used without the *no_init* option. In all other cases, only the `LINE_FIRM_BOOT_ROM_*` macros will be available.

LINE_FIRM_NUM_DSPS *(lp)*

Returns the number of DSPs with version information available. Each DSP provides its own version information that can be independently accessed.

LINE_FIRM_DSP_MAJOR *(lp,n)*

Returns the major portion of the *n*th DSP firmware version number.

The value of *n* ranges from 0 to `LINE_FIRM_NUM_DSPS` (*lp*) –1.

LINE_FIRM_DSP_MIDDLE *(lp,n)*

Returns the middle portion of the *n*th DSP firmware version number.

The value of *n* ranges from 0 to `LINE_FIRM_NUM_DSPS` (*lp*) –1.

LINE_FIRM_DSP_MINOR *(lp,n)*

Returns the minor portion of the *n*th DSP firmware version number.

The value of *n* ranges from 0 to `LINE_FIRM_NUM_DSPS` (*lp*) –1.

LINE_FIRM_DSP_BUILD *(lp,n)*

Returns the build number of the *n*th DSP firmware version.

The value of *n* ranges from 0 to `LINE_FIRM_NUM_DSPS` (*lp*) –1.

LINE_FIRM_DSP_AUTO_NUM *(lp,n)*

Returns the *n*th DSP firmware "auto number". This value is used internally and refers to internal build procedures.

LINE_FIRM_DSP_DATE *(lp,n)*

Returns the date of the *n*th DSP firmware version. The date is a character string in the form M/D/Y (for example, 1/31/02).

The value of *n* ranges from 0 to `LINE_FIRM_NUM_DSPS` (*lp*) –1.

LINE_FIRM_DSP_COMMENT *(lp,n)*

Returns comments about the *n*th DSP firmware version.

LINE_FIRM_MAJOR *(lp)*

Returns the major portion of the firmware version number. For example, if the version number is 6.3 or 6.4.3, this macro returns 6.

LINE_FIRM_MIDDLE *(lp)*

Returns the middle portion of the firmware version number. For example, if the version number is 6.4.3, this macro returns 4.

LINE_FIRM_MINOR *(lp)*

Returns the minor portion of the firmware version number. For example, if the version number is 6.3 or 6.4.3, this macro returns 3.

LINE_FIRM_DATE *(lp)*

Returns the date stored in the firmware that indicates when the firmware was developed. The date is a character string in the form M/D/Y (for example, 1/31/02).

LINE_FIRM_CHECKSUM *(lp)*

Returns an unsigned short that contains the firmware checksum.

This macro returns 0.

LINE_FIRM_CHK_OK *(lp)*

Returns a value generated by the module that indicates whether or not the computed firmware checksum matches the stored firmware checksum.

1 Checksums match.

0 Checksums do not match.

This macro returns 1.

LINE_FIRM_DOWNLOADED *(lp)*

For compatibility only.

Returns a nonzero value if firmware was successfully downloaded to the module since the last hardware reset (power up).

This macro returns 1.

LINE_FIRM_ID *(lp)*

Returns a character string that contains the default local ID stored in the firmware. This ID is often used to verify that customized firmware is in use.

This macro returns an empty string.

LINE_FIRM_TYPE *(lp)*

For compatibility only. Returns a number that identifies the board type stored in the firmware. Values are:

1 TR111MC

2 TR111WG

4 TR112

5 TR112DID

6 TR112T1

7 TruFax®

8 TruFaxAEB

114 TR114 and all other Dialogic® Brooktrout® boards

4 - Configuration

This chapter describes functions to initialize, configure and query telephony and network ports and also functions to configure the interconnections between channels and telephony or network ports.

The Bfv API provides functions that allow you to get the current information about the telephony configuration, reset the telephony state, and save telephony parameters to Non-Volatile RAM (NVRAM).

You can also establish a connection between source and destination telephony resources; get information about the connections, their ports and classes; and clear all switching connections for a module.

Configuration Files

The Bfv API uses several configuration files that let you configure the Bfv API and driver, call control, and country-specific parameters. These files are stored in the directory *brooktrout/boston/config* and are described below:

- The user-defined configuration file

  A file that contains configuration parameters for the Bfv API and driver. A sample of this file, called *btcall.cfg*, is provided with the software, but you can write your own or modify/rename the existing one. Many of the sample applications (see *Sample Applications and Utilities* in your *Developer Guide, or Appendix A* of *Volume 6*) use *btcall.cfg*.

- The call control configuration file

  A user-supplied file that contains call control configuration parameters. Several samples of this file are provided with the software. One sample is called *callctrl.cfg*, while others have names that specify the type of telephony interface. See the directory *BrooktroutBoston/config/samples.cfg* for the names of the files, or *Appendix A* of *Volume 6*.

- The *BT_CPARM.CFG* file.

  A read-only file that contains country-specific parameters. See *Appendix G*, *Volume 6* for more information.

Function Summary

Table 3 provides a brief summary of the functions used for configuration.

Table 3. Configuration Function Summary

Function	Purpose	Page
BfvCallSWClearConns	Clears all call switching connections on the current module.	*119*
BfvCallSWConnect	Forms a connection between specified source and destination telephony resources.	*121*
BfvCallSWGetConns	Retrieves and returns information about established call switching connections on the current module.	*142*
BfvCallSWGetInfo	Retrieves and returns information about the connectable port classes and units.	*146*
BfvNetworkConfigGet	Returns network configuration information about a single specified interface or trunk.	*149*
args.unit	Performs network configuration on a collection of interfaces and trunks.	*154*
BfvNetworkQuery	Retrieves statistics about the specified Ethernet unit.	*158*
BfvTelephGetInfo	Returns network configuration information about a single specified interface or trunk.	*163*
BfvTelephReset	Modifies the telephony state of the current module to permit performing high level configuration again using *BfvCallCtrlInit*.	*165*
BfvTelephSave	Saves already configured telephony parameters to NVRAM. This enables boards to power up and initialize with legal configurations for the environment.	*167*

BfvCallSWClearConns

Purpose

Clears all call switching connections on the current module.

This function does not support hardware without an H.100 connector. The SR140 also does not support this function.

Syntax

```
int
BfvCallSWClearConns(lp, args)
                    BTLINE                    lp;
                    struct args_tel_ctrl_call_sw*args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value:

0	Success.
<0	Error.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function clears all switching connections between source and destination resources on the current module.

See Also

BfvCallSWConnect, *BfvCallSWGetInfo*

Example

```
BTLINE *lp;
struct args_tel_ctrl_call_sw args;

/* Clear all connections on module */
BT_ZERO(args);
BfvCallSWClearConns(lp, &args);
```

BfvCallSWConnect

Purpose

Forms a connection between specified source and destination resources on the current module.

This function does not support hardware without an H.100 connector. The SR140 also does not support this function.

Syntax

```
int
BfvCallSWConnect  (lp, args)
                     BTLINE                    *lp;
                     struct args_tel_ctrl_call_sw*args;
```

The structure contains the following fields.

Input Fields

```
unsigned conn_mode;
unsigned src_port_class;
unsigned src_port_unit;
unsigned src_stream;
unsigned src_slot;
unsigned dest_port_class;
unsigned dest_port_unit;
unsigned dest_stream;
unsigned dest_slot;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.conn_mode

Type of connection to be established or an indication to terminate a previously established connection. Valid values are:

CALL_SW_DISCONNECT_DEF	0x00
CALL_SW_TRANSMIT_ONLY_DEF	0x01
CALL_SW_RECEIVE_ONLY_DEF	0x02
CALL_SW_FULL_DUPLEX_DEF	0x03
CALL_SW_DUPLEX_AND_SIGNALING_DEF	0x07

args.src_port_class

Port class of the source resource. Set to one of the following:

CALL_SW_PORT_CHANNEL_DEF	0x00
CALL_SW_PORT_BUS_DEF	0x0E
CALL_SW_PORT_NETWORK_DEF	0x0F
CALL_SW_PORT_WILD_DEF	0x10

CALL_SW_PORT_WILD_DEF can be used only when *args.conn_mode* is CALL_SW_DISCONNECT_DEF, indicating that all connections to the specified destination are to be disconnected.

args.src_port_unit

Source resource port unit number.

If *args.src_port*_class is CALL_SW_PORT_CHANNEL_DEF: this indicates the logical channel number. The first channel on each module is logical channel number 2, the second is logical channel 3, and so forth. To determine the logical channel value of a channel with an attached line pointer, use the LINE_SRC_ADDR macro's *mm_bChannel* field (see *Low-Level Macros on page 90*).

If *args.src_port_class* is CALL_SW_PORT_NETWORK_DEF: this indicates the physical interface. For T1/E1 and BRI, this is the span/trunk (0-based). For analog, this is the number of the analog line (0-based).

Example:

24 channel T1	0
4 channel BRI	0-1
4 port analog	0-3
8 port analog	0-8

`args.src_stream`

Source resource stream number.

If *args.src_port_class* is `CALL_SW_PORT_BUS_DEF`:

For H.100 the valid range is 0-31.

Otherwise, must be zero.

`args.src_slot`

Source resource time slot number.

If *args.src_port* is `CALL_SW_PORT_BUS_DEF`:

For H.100 the valid ranges are:

2 MHz bus speed0-31
4 MHz bus speed0-63
8 MHz bus speed0-127

Otherwise, must be zero.

`args.dest_port_class`

Port class of the destination resource. Set to one of the following:

`CALL_SW_PORT_CHANNEL_DEF`	0x00
`CALL_SW_PORT_BUS_DEF`	0x0E
`CALL_SW_PORT_NETWORK_DEF`	0x0F
`CALL_SW_PORT_WILD_DEF`	0x10

`CALL_SW_PORT_WILD_DEF` can be used only when *args.conn_mode* is `CALL_SW_DISCONNECT_DEF`, indicating that all connections to the specified source are to be disconnected.

`args.dest_port_unit`

Destination resource port unit number.

If *args.dest_port*_class is `CALL_SW_PORT_CHANNEL_DEF`: this indicates the logical channel number. The first channel on each module is logical channel number 2, the second is logical channel 3, and so forth. To determine the logical channel value of a channel with an attached line pointer, use the `LINE_DEST_ADDR` macro's *mm_bChannel* field (see *Low-Level Macros on page 90*).

If *args.dest_port_class* is `CALL_SW_PORT_NETWORK_DEF`: this indicates the physical interface. For T1/E1 and BRI, this is the span/trunk (0-based). For analog, this is the number of the analog line (0-based).

Example:

24 channel T1	0
4 channel BRI	0-1
4 port analog	0-3
8 port analog	0-8

Otherwise, must be zero.

`args.dest_stream`

Destination resource stream number.

If *args.dest_port_class* is `CALL_SW_PORT_BUS_DEF`:

For H.100 the valid range is 0-31.

Otherwise, must be zero.

`args.dest_slot`

Destination resource time slot number.

If *args.dest_port* is `CALL_SW_PORT_BUS_DEF`:

For H.100 the valid ranges are:

2 MHz bus speed0-31
4 MHz bus speed0-63
8 MHz bus speed0-127

Otherwise, must be zero.

Output

Return value:

0	Success
>0	Operational error reported by firmware
<0	Other error

`CALL_SW_GENERIC_FAILURE_DEF`	0x01
`CALL_SW_UNKNOWN_CONN_MODE_DEF`	0x02
`CALL_SW_UNKNOWN_PORT_TYPE_DEF`	0x03
`CALL_SW_INVALID_PORT_TYPE_DEF`	0x04
`CALL_SW_UNKNOWN_PORT_UNIT_DEF`	0x05
`CALL_SW_INVALID_PORT_UNIT_DEF`	0x06
`CALL_SW_INVALID_PORT_STREAM_DEF`	0x07
`CALL_SW_INVALID_STREAM_SLOT_DEF`	0x08
`CALL_SW_SLOT_ALREADY_ASSIGNED_DEF`	0x09
`CALL_SW_SLOT_NOT_ASSIGNED_DEF`	0x0A

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The resources can each represent a channel, or any timeslot on any telephony bus or network entity.

Once a connection is established, it remains established until explicitly disconnected. It is important that applications terminate connections prior to exiting, if the connection is not permanent.

Initialize T1/E1 controllers that are to be used prior to executing *BfvCallSWConnect* calls. If you initialize T1/E1 controllers after *BfvCallSWConnect* calls have been made, the Bfv API destroys the connections previously made.

To disconnect a full-duplex connection, you must treat the connection as two simplex connections. Therefore, you will need to call *BfvCallSWConnect* two times (once for each direction). The arguments for each call will be the same, except that the source and destination connection points will be reversed.

To disconnect a `TRANSMIT_ONLY` switch connection, reverse the source and destination fields for the `DISCONNECT` call. In other words, when disconnecting, set the source field to the endpoint that is receiving the data.

See Also

BfvCallSWClearConns, *BfvCallSWGetInfo*

Example

```
BTLINE *lp;
struct args_tel_ctrl_call_sw args;

/* Connect logical channel 2 to stream 0, slot 0
   of the first network unit. */
BT_ZERO(args);
args.conn_mode = CALL_SW_DUPLEX_AND_SIGNALING_DEF;
args.src_port_class = CALL_SW_PORT_CHANNEL_DEF;
args.src_port_unit = 2;
args.src_stream = 0;
args.src_slot = 0;
args.dest_port_class = CALL_SW_NETWORK_DEF;
args.dest_port_unit = 0;
args.dest_stream = 0;
args.dest_slot = 0;
BfvCallSWConnect(lp, &args);
```

BFVCallSWConnectIP

Purpose

Forms a connection between specified channel and IP address and port.

This function does not support hardware without a network interface or configuration where Bfv manages the PSTN or IP calls. Use this function only when integrating third party call control stacks. Refer to the section *Using Third Party IP Stacks* in the *Dialogic® Brooktrout® Fax Products SDK Developer's Guide.*

Syntax

```
int
BfvCallSWConnectIP(lp, args)
                        BTLINE                    *lp;
                        struct args_tel_ctrl_call_sw*args;
```

The structure contains the following fields.

Input Fields

```
unsigned          conn_mode;
unsigned          src_port_unit;
unsigned          dest_port_class;


struct bt_sockaddr_storage destOptions.RTPopts.localRTPNgAddr;
struct bt_sockaddr_storage destOptions.RTPopts.localRTCPNgAddr;
struct bt_sockaddr_storage destOptions.RTPopts.remoteRTPNgAddr;
struct bt_sockaddr_storage destOptions.RTPopts.remoteRTCPNgAddr;
struct bt_sockaddr_storage destOptions.UDPTLopts.localNgAddr;
struct bt_sockaddr_storage destOptions.UDPTLopts.remoteNgAddr;
unsigned destOptions.UDPTLopts.t38_version;
unsigned destOptions.UDPTLopts.t38_max_bit_rate;
unsigned destOptions.UDPTLopts.t38_fax_fill_bit_removal;
unsigned destOptions.UDPTLopts.t38_fax_transcoding_MMR;
unsigned destOptions.UDPTLopts.t38_fax_transcoding_JBIG;
unsigned destOptions.UDPTLopts.t38_fax_rate_management;
unsigned destOptions.UDPTLopts.t38_fax_max_buffer;
unsigned destOptions.UDPTLopts.t38_fax_max_datagram_recv;
unsigned destOptions.UDPTLopts.t38_fax_udp_EC;
unsigned destOptions.UDPTLopts.t38_fax_max_datagram_send;
```

```
unsigned destOptions.UDPTLopts.t38_UDPTL_redundancy_depth_control;
unsigned destOptions.UDPTLopts.t38_t30_fast_notify;
```

Output Fields

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.conn_mode

Type of connection to be established or an indication to terminate a previously established connection.

Valid values are:

CALL_SW_RECEIVE_ONLY_DEF

CALL_SW_FULL_DUPLEX_DEF

CALL_SW_DISCONNECT_MODE_DEF

args.src_port_unit

Source resource port unit number.

This indicates the logical channel number. The first channel on each module is logical channel number 2, the second is logical channel 3, and so forth. To determine the logical channel value of a channel with an attached line pointer, use the LINE_SRC_ADDR macro's mm_bChannel field

args.dest_port_class

This indicates the type of connection RTP or UDPTL.

Valid values are:

CALL_SW_PORT_RTP_DEF

CALL_SW_PORT_UDPTL_DEF

args.destOptions.RTPopts.localRTPNgAddr

This contains the local RTP address which is stored in a Next Generation struct bt_sockaddr_storage structure and can be either an IPv4 address or an IPv6 address. Depending on the type of IP

address being specified, this field must be cast to an appropriate IP address structure type (either struct bt_sockaddr_in for IPv4 addresses or struct bt_sockaddr_in6 for IPv6 addresses) and then the IP address specific settings can be specified.

The layout of these structures is provided below:

```
struct bt_sockaddr_storage
{
    uint8_t        __ss_len;
    bt_sa_family_t __ss_family;
    char           __ss_pad1[_BT_SS_PAD1SIZE];
    int64_t        __ss_align;
    char           __ss_pad2[_BT_SS_PAD2SIZE];
};

struct bt_sockaddr_in
{
    short sin_family;
    unsigned short sin_port;
    struct bt_in_addr sin_addr;
    char sin_zero[8];
};

struct bt_sockaddr_in6
{
    short          sin6_family;
    unsigned short sin6_port;
    unsigned int   sin6_flowinfo;
    struct bt_in6_addr sin6_addr;
    unsigned int   sin6_scope_id;
};
```

The names and types of the IPv4 and IPv6 specific fields that can be populated differ according to the IP address type as illustrated in the following table:

IP Address Field	IPv4Address	IPv6Address
Family Type	sin_family	sin6_family
Port Number	sin_port	sin6_port
IP Address	sin_addr	sin6_addr

IP Address Field	IPv4Address	IPv6Address
Scope ID	N/A	sin6_scope_id

A detailed description of each of these fields is provided below:

Family Type

This indicates the type of local RTP addressing.

Valid values are:

TELE_CTRL_AF_INET_DEF (IPv4 Address Family)
TELE_CTRL_AF_INET6_DEF(IPv6 Address Family)

Port Number

This indicates the local RTP port number.

IP Address

This indicates the local RTP IP address and can be either an IPv4 or IPv6 address. IPv4 addresses can be specified as a four byte integer where

Class A is byte 4
Class B is byte 3
Class C is byte 2
Class D is byte 1
Example: 10.128.100.100 would be 0x0A806464

IPv4 addresses are assigned to a struct bt_in_addr structure. The layout of this structure is provided below:

```
struct bt_in_addr
{
    union {
        struct {
            unsigned char s_b1, s_b2, s_b3, s_b4;
        } S_un_b;
        struct {
            unsigned short s_w1, s_w2;
        } S_un_w;
        unsigned int S_addr;
    } S_un;
};
```

In contrast to IPv4, IPv6 addresses are specified as 16 bytes. IPv6 addresses are stored in a struct bt_in6_addr structure, the layout of which is provided below:

```
struct bt_in6_addr
{
    union
    {
        unsigned char _S6_u8[16];
        unsigned int  _S6_u32[4];
        uint64_t      _S6_u64[2];
    } _S6_un;
};
```

In contrast to IPv4, IPv6 addresses are specified as 16 bytes. IPv6 addresses are stored in a struct bt_in6_addr structure, the layout of which is provided below:

```
struct bt_in6_addr
{
    union
    {
        unsigned char _S6_u8[16];
        unsigned int  _S6_u32[4];
        uint64_t      _S6_u64[2];
    } _S6_un;
};
```

Scope ID

This indicates a set of interfaces as appropriate for the scope of the specified IP address. For link-local IP addresses, this would be an interface index.

Note: This field is only applicable to IPv6 addresses.

In addition to populating the IP address fields defined above, the args.destOptions.RTPopts.localRTPNgAddr.__ss_len field must be populated with the size of the structure used to specify the actual IP address settings. An example of how to populate these fields is shown below:

IPv4 Example

```
unsigned short port = 56008;
```

```
struct bt_sockaddr_in *pIPv4Addr;
char *szIPv4Addr =;
args.destOptions.RTPopts.localRTPNgAddr.__ss_len =
sizeof(struct bt_sockaddr_in);
pIPv4Addr = (struct bt_sockaddr_in
*)&args.destOptions.RTPopts.localRTPNgAddr.__ss_family;
pIPv4Addr->sin_family = TELE_CTRL_AF_INET_DEF;
pIPv4Addr->sin_port   = port;

/* Specify the IP address "10.128.100.55" */
pIPv4Addr->sin_addr.S_un.S_un_b.s_b4 = 10;
pIPv4Addr->sin_addr.S_un.S_un_b.s_b3 = 128;
pIPv4Addr->sin_addr.S_un.S_un_b.s_b2 = 100;
pIPv4Addr->sin_addr.S_un.S_un_b.s_b1 = 55;
```

IPv6 Example

```
unsigned int scope_id = 3;
unsigned short port = 56008;
struct bt_sockaddr_in6 *pIPv6Addr;

args.destOptions.RTPopts.localRTPNgAddr.__ss_len =
sizeof(struct bt_sockaddr_in6);
pIPv6Addr = (struct bt_sockaddr_in6
*)&args.destOptions.RTPopts.localRTPNgAddr.__ss_family;
pIPv6Addr->sin6_family   = TELE_CTRL_AF_INET6_DEF;
pIPv6Addr->sin6_port     = port;
pIPv6Addr->sin6_scope_id = scope_id;

/* Specify the IPv6 address
"fe80:0000:0000:0000:0211:25ff:fed3:fdf2" */
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 0] = 0xFE;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 1] = 0x80;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 2] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 3] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 4] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 5] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 6] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 7] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 8] = 0x02;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 9] = 0x11;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[10] = 0x25;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[11] = 0xFF;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[12] = 0xFE;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[13] = 0xD3;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[14] = 0xFD;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[15] = 0xF2;
```

`args.destOptions.RTPopts.localRTCPNgAddr`

This contains the local RTCP address which is stored in a Next Generation struct bt_sockaddr_storage structure and can be either an IPv4 address or an IPv6 address. Refer to args.destOptions.RTPopts.localRTPNgAddr for information describing the supported fields within this structure.

`args.destOptions.RTPopts.remoteRTPNgAddr`

This contains the remote RTP address which is stored in a Next Generation struct bt_sockaddr_storage structure and can be either an IPv4 address or an IPv6 address. Refer to args.destOptions.RTPopts.localRTPNgAddr for information describing the supported fields within this structure.

`args.destOptions.RTPopts.remoteRTCPNgAddr`

This contains the remote RTCP address which is stored in a Next Generation struct bt_sockaddr_storage structure and can be either an IPv4 address or an IPv6 address. Refer to args.destOptions.RTPopts.localRTPNgAddr for information describing the supported fields within this structure.

`args.destOptions.UDPTLopts.localNgAddr`

This contains the local UDPTL address which is stored in a Next Generation struct bt_sockaddr_storage structure and can be either an IPv4 address or an IPv6 address. Refer to args.destOptions.RTPopts.localRTPNgAddr for information describing the supported fields within this structure.

`args.destOptions.UDPTLopts.remoteNgAddr`

This contains the remote UDPTL address which is stored in a Next Generation struct bt_sockaddr_storage structure and can be either an IPv4 address or an IPv6 address. Refer to args.destOptions.RTPopts.localRTPNgAddr for information describing the supported fields within this structure.

`args.destOptions.UDPTLopts.t38_version`

Controls the maximum T.38 ASN.1 version the IP Call Control offers or accepts from a remote party. Versions 0, 1, and 2 support a maximum bit rate of 14,400 bps. Version 3 supports V.34. with a maximum bit rate of 33,600 bps.

Valid values are:

0 - 3

`args.destOptions.UDPTLopts.t38_max_bit_rate`

Specifies the negotiated value that defines the maximum bit rate for fax packetization onto the network.

Valid values are:

CALL_SW_BIT_RATE_2400_DEF
CALL_SW_BIT_RATE_4800_DEF
CALL_SW_BIT_RATE_7200_DEF
CALL_SW_BIT_RATE_9600_DEF
CALL_SW_BIT_RATE_12000_DEF
CALL_SW_BIT_RATE_14400_DEF
CALL_SW_BIT_RATE_16800_DEF
CALL_SW_BIT_RATE_19200_DEF
CALL_SW_BIT_RATE_21600_DEF
CALL_SW_BIT_RATE_24000_DEF
CALL_SW_BIT_RATE_26400_DEF
CALL_SW_BIT_RATE_28800_DEF
CALL_SW_BIT_RATE_31200_DEF
CALL_SW_BIT_RATE_33600_DEF

`args.destOptions.UDPTLopts.t38_fax_fill_bit_removal`

Specifies whether the Bfv API can remove or insert fill bits to reduce the bandwidth of the transport mechanism.

Valid values are:

0 Indicates that the Bfv API does not support the capability.

1 Indicates that the Bfv API can remove or insert fill bits.

`args.destOptions.UDPTLopts.t38_fax_transcoding_MMR`

Specifies whether the Bfv API can convert to and from MMR fax compression to reduce the bandwidth of the transport mechanism when using a reliable transport (for example, TCP).

Valid values are:

0 Indicates that the Bfv API does not support the capability.

1 Indicates that the Bfv API can convert MMR compression.

`args.destOptions.UDPTLopts.t38_fax_transcoding_JBIG`

Specifies whether the Bfv API can convert to and from JBIG fax images to reduce the bandwidth of the transport mechanism when using a reliable transport (for example, TCP).

Valid values are:

0 Indicates that the Bfv API does not support the capability.

1 Indicates that the Bfv API can convert JBIG fax images.

args.destOptions.UDPTLopts.t38_fax_rate_management

Specifies a value that identifies the data rate management method of the transport.

Valid values are:

CALL_SW_LOCAL_TCF_DEF

Indicates that the transport uses the local training check frame (TCF) data rate management type (not supported).

CALL_SW_TRANSFERRED_TCF_DEF

Indicates that the transport uses the transferred training check frame (TCF) data rate management type.

args.destOptions.UDPTLopts.t38_fax_max_buffer

Maximum fax buffer.

Valid value: 200

args.destOptions.UDPTLopts.t38_fax_max_datagram_recv

Maximum datagram for receive.

Valid value: 72

args.destOptions.UDPTLopts.t38_fax_udp_EC

Specifies a value that identifies the error correction method of the T.38 fax transport.

Valid values are:

CALL_SW_FEC_DEF The transport uses the T.38 user datagram protocol (UDP) forward error correction (FEC) method (not supported).

CALL_SW_REDUNDANCY_DEF The transport uses the T.38 UDP redundancy error correction method.

args.destOptions.UDPTLopts.t38_fax_max_datagram_send

Maximum datagram for transmit.

Valid value: 72

args.destOptions.UDPTLopts.t38_UDPTL_redundancy_depth_control

Specifies a value that defines the number of prior messages to include as redundancy messages in a transmitted UDPTL packet carrying signal information (FSK signals).

Valid values are:

0 - 5 Specifies a number value defining how many prior messages to include as redundancy messages in a packet carrying control data.

args.destOptions.UDPTLopts.t38_t30_fast_notify

Specifies whether the transport signals the beginning of T.30 by means of a zero-length data field or uses a T.30 indicator value.

Valid values are:

0 Indicates that the T.38 fax transport uses a zero-length data field to signal the beginning of T.30.

1 Indicates that the transport uses a T30_INDICAT

Output

Return value:

0 Success

>0 Operational error reported by firmware

<0 Other error

args.res

A RES structure containing status information. For more information, see Volume 6, Appendix B, Result Structures.

CALL_SW_GENERIC_FAILURE_DEF	0x01
CALL_SW_UNKNOWN_CONN_MODE_DEF	0x02
CALL_SW_UNKNOWN_PORT_TYPE_DEF	0x03
CALL_SW_INVALID_PORT_TYPE_DEF	0x04
CALL_SW_UNKNOWN_PORT_UNIT_DEF	0x05
CALL_SW_INVALID_PORT_UNIT_DEF	0x06
CALL_SW_INVALID_PORT_STREAM_DEF	0x07
CALL_SW_INVALID_STREAM_SLOT_DEF	0x08
CALL_SW_SLOT_ALREADY_ASSIGNED_DEF	0x09

CALL_SW_SLOT_NOT_ASSIGNED_DEF	0x0A
CALL_SW_NO_T38_LICENSE_DEF	0x0B
CALL_SW_PORT_RTP_DEF	0x013
CALL_SW_IP_PORT_INUSE_DEF	0x014
CALL_SW_INVALID_CONN_MODE_DEF	0x15
CALL_SW_IP_PORT_REUSE_DEF	0x16

Details

The source resource needs to be a channel.

Once a connection is established, it remains established until explicitly disconnected. It is important that applications terminate connections prior to exiting.

To disconnect a connection, you must OR CALL_SW_DISCONNECT_MODE_DEF with the parameters used to set the connection. For example to disconnect a full duplex connection set conn_mode to CALL_SW_FULL_DUPLEX_DEF | CALL_SW_DISCONNECT_MODE_DEF.

With the introduction of the Next Generation IP address fields to the *RTP_options and UDPTL_options* structures, the following IPv4-only fields are deprecated and are included in the structures for backwards compatibility only.

args.destOptions.RTPopts.localRTPAddr

args.destOptions.RTPopts.remoteRTPAddr

args.destOptions.RTPopts.localRTCPAddr

args.destOptions.RTPopts.remoteRTCPAddr

args.destOptions.UDPTLopts.localAddr

args.destOptions.UDPTLopts.remoteAddr

Application developers should use the Next Generation IP address fields to specify IPv4 and IPv6 addresses instead of these fields.

Example

```
BTLINE *lp;
struct args_tel_ctrl_call_sw args;
struct bt_sockaddr_in6 *pIPv6Addr;
char *szLocalAddr  = "fe80::211:25ff:fed3:fdf2";
char *szRemoteAddr = "fe80::204:e2ff:fe3e:c089";
```

```
/* Connect logical channel 2 to RTP */
BT_ZERO(args);
args.conn_mode = CALL_SW_FULL_DUPLEX_DEF;
args.src_port_unit = 2;
args.dest_port_class = CALL_SW_PORT_RTP_DEF;

/*
 * RTP local IP Address [fe80::211:25ff:fed3:fdf2]:56008
 * and RTCP local IP Address
[fe80::211:25ff:fed3:fdf2]:56009
 * are used in the example below.
 */
args.destOptions.RTPopts.localRTPNgAddr.__ss_len =
                                          sizeof(struct
bt_sockaddr_in6);
pIPv6Addr = (struct bt_sockaddr_in6 *)

&args.destOptions.RTPopts.localRTPNgAddr.__ss_family;
pIPv6Addr->sin6_family   = TELE_CTRL_AF_INET6_DEF;
pIPv6Addr->sin6_port     = 56008;
pIPv6Addr->sin6_scope_id = 3; /* Use interface 3 for this
IP address */

/* Specify the IPv6 address
"fe80:0000:0000:0000:0211:25ff:fed3:fdf2" */
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 0] = 0xFE;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 1] = 0x80;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 2] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 3] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 4] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 5] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 6] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 7] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 8] = 0x02;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 9] = 0x11;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[10] = 0x25;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[11] = 0xFF;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[12] = 0xFE;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[13] = 0xD3;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[14] = 0xFD;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[15] = 0xF2;


/* Configure the local RTCP IP address settings */
args.destOptions.RTPopts.localRTCPNgAddr.__ss_len =
                                          sizeof(struct
bt_sockaddr_in6);
pIPv6Addr = (struct bt_sockaddr_in6 *)
```

```
&args.destOptions.RTPopts.localRTCPNgAddr.__ss_family;
pIPv6Addr->sin6_family   = TELE_CTRL_AF_INET6_DEF;
pIPv6Addr->sin6_port     = 56009;
pIPv6Addr->sin6_scope_id = 3; /* Use interface 3 for this
IP address */

/* Specify the IPv6 address
"fe80:0000:0000:0000:0211:25ff:fed3:fdf2" */
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 0] = 0xFE;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 1] = 0x80;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 2] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 3] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 4] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 5] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 6] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 7] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 8] = 0x02;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 9] = 0x11;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[10] = 0x25;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[11] = 0xFF;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[12] = 0xFE;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[13] = 0xD3;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[14] = 0xFD;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[15] = 0xF2;

/*
 * RTP remote IP Address [fe80::204:e2ff:fe3e:c089]:56008
 * and RTCP remote IP Address
[fe80::204:e2ff:fe3e:c089]:56009
 * are used in the example below.
 */
args.destOptions.RTPopts.remoteRTPNgAddr.__ss_len =
                                          sizeof(struct
bt_sockaddr_in6);
pIPv6Addr = (struct bt_sockaddr_in6 *)

&args.destOptions.RTPopts.remoteRTPNgAddr.__ss_family;
pIPv6Addr->sin6_family   = TELE_CTRL_AF_INET6_DEF;
pIPv6Addr->sin6_port     = 56008;
pIPv6Addr->sin6_scope_id = 3; /* Use interface 3 for this
IP address */


pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 0] = 0xFE;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 1] = 0x80;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 2] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 3] = 0x00;
```

```
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 4] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 5] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 6] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 7] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 8] = 0x02;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 9] = 0x04;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[10] = 0xE2;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[11] = 0xFF;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[12] = 0xFE;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[13] = 0x3E;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[14] = 0xC0;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[15] = 0x89;


/* Configure the local RTCP IP address settings */
args.destOptions.RTPopts.remoteRTCPNgAddr.__ss_len =
                                          sizeof(struct
bt_sockaddr_in6);
pIPv6Addr = (struct bt_sockaddr_in6 *)

&args.destOptions.RTPopts.remoteRTCPNgAddr.__ss_family;
pIPv6Addr->sin6_family   = TELE_CTRL_AF_INET6_DEF;
pIPv6Addr->sin6_port     = 56009;
pIPv6Addr->sin6_scope_id = 3; /* Use interface 3 for this
IP address */

/* Specify the IPv6 address
"fe80:0000:0000:0000:0204:e2ff:fe3e:c089" */
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 0] = 0xFE;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 1] = 0x80;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 2] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 3] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 4] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 5] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 6] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 7] = 0x00;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 8] = 0x02;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[ 9] = 0x04;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[10] = 0xE2;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[11] = 0xFF;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[12] = 0xFE;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[13] = 0x3E;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[14] = 0xC0;
pIPv6Addr->sin6_addr.in6_u.u6_addr8[15] = 0x89;


rc = BfvCallSWConnectIP(lp, &args);
```

```
switch (rc)
{
   case CALL_SW_SUCCESS_DEF:
// Port was available and reserved
break;
   case CALL_SW_IP_PORT_INUSE_DEF:
// Try again with a different local port
break;
   case CALL_SW_NO_T38_LICENSE_DEF:
// Out of T.38 resources
break;
   default:
// Most likely an API error
if (rc > 0)
{
// error code reported from attempt before
// the call was sent to the FW
// Expect:
   // args.res.status = BT_STATUS_ERROR;
// args.res.line_status = APIERR_BADPARAMETER;
}

if (rc < 0)
{
// other error usually sent by the FW
```

BfvCallSWGetConns

Purpose

Retrieves and returns information about established call switching connections on the current module.

This function does not support hardware without an H.100 connector. The SR140 also does not support this function.

Syntax

```
int
BfvCallSWGetConns (lp, args)
                      struct args_tel_ctrl_call_sw*args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

```
unsigned conn_mode;
unsigned src_port_class;
unsigned src_port_unit;
unsigned src_stream;
unsigned src_slot;
unsigned src_port_type;
unsigned dest_port_class;
unsigned dest_port_unit;
unsigned dest_stream;
unsigned dest_slot;
unsigned dest_port_type;
RES res;
```

Input

lp

Pointer to the `BTLINE` structure.

Output

Return value:

1	Connection information is returned.
0	No more connection information.
–1	Error.

args.conn_mode

Type of connection to be established or an indication to terminate a previously established connection. Valid values are:

CALL_SW_TRANSMIT_ONLY_DEF	0x01
CALL_SW_FULL_DUPLEX_DEF	0x03

args.src_port_class

Port class of the source resource. Set to one of the following:

CALL_SW_PORT_CHANNEL_DEF	0x00
CALL_SW_PORT_BUS_DEF	0x0E
CALL_SW_PORT_NETWORK_DEF	0x0F

args.src_port_unit

Source resource port unit number for the source port class. If *args.src_port_class* is CALL_SW_PORT_CHANNEL_DEF, then the value indicates the logical channel number. The first channel on each module is logical channel number 2, the second is logical channel 3, and so forth.

args.src_stream

Source resource stream number.

args.src_slot

Source resource time slot number.

args.src_port_type

A value specifying the port type of the source port. Valid values are:

CALL_SW_PORT_CHANNEL_DEF	0x00
CALL_SW_PORT_H100_DEF	0x05
CALL_SW_PORT_T1_DEF	0x06
CALL_SW_PORT_E1_DEF	0x07
CALL_SW_PRI_T1_DEF	0x08
CALL_SW_PRI_E1_DEF	0x09
CALL_SW_AT_MODEM_DEF	0x0A
CALL_SW_PORT_BRI_DEF	0x0B

CALL_SW_PORT_ANALOG_LOOP_START_DEF	0x0C
CALL_SW_PORT_ANALOG_DID_DEF	0x0D

args.dest_port_class

Port class of the destination resource. Set to one of the following:

CALL_SW_PORT_CHANNEL_DEF	0x00
CALL_SW_PORT_BUS_DEF	0x0E
CALL_SW_PORT_NETWORK_DEF	0x0F

args.dest_port_unit

Destination resource port unit number for the destination port class. If *args.dest_port_class* is CALL_SW_PORT_CHANNEL_DEF, then the value indicates the logical channel number. The first channel on each module is logical channel number 2, the second is logical channel 3, etc.

args.dest_stream

Destination resource stream number.

args.dest_slot

Destination resource time slot number.

args.dest_port_type

A value specifying the port type of the destination port. Valid values are:

CALL_SW_PORT_CHANNEL_DEF	0x00
CALL_SW_PORT_H100_DEF	0x05
CALL_SW_PORT_T1_DEF	0x06
CALL_SW_PORT_E1_DEF	0x07
CALL_SW_PRI_T1_DEF	0x08
CALL_SW_PRI_E1_DEF	0x09
CALL_SW_AT_MODEM_DEF	0x0A
CALL_SW_PORT_BRI_DEF	0x0B
CALL_SW_PORT_ANALOG_LOOP_START_DEF	0x0C
CALL_SW_PORT_ANALOG_DID_DEF	0x0D

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function makes the assumption that connections will not change for the duration of the query procedure. Use this function for only one application session at a time.

Each call returns information about one connection. Call repeatedly until the return value = 0 (no more information).

The procedure is quite time consuming for the firmware and should only be used for debugging or explicit status requirements. Don't use the procedure on a frequent, regular basis.

When *args.src_port_class* or *args.dest_port_class* is CALL_SW_PORT_CHANNEL_DEF, logical channel values must be used in the corresponding *port_unit* input field. To determine the logical channel value of a channel with an attached line pointer, the LINE_DEST_ADDR macro's *mm_bChannel* field can be used (see *Low-Level Macros on page 90*).

Note: A receive connection between two given points is identical to a transmit connection between those same points with the source and destination reversed.

See Also

BfvCallSWConnect

Example

See the *connlist.c* sample application in the *bapp.src* directory.

BfvCallSWGetInfo

Purpose

Retrieves and returns information about the connectable port classes and units on the current module.

This function does not support hardware without an H.100 connector. The SR140 also does not support this function.

Syntax

```
int
BfvCallSWGetInfo  (lp, args)
                     BTLINE                    *lp;
                     struct args_tel_ctrl_call_sw*args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

```
unsigned src_port_class;
unsigned src_port_unit;
unsigned src_stream;
unsigned src_slot;
unsigned port_type;
RES res;
```

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value:

1	Unit information is returned.
0	No more unit information.
<0	Error.

args.src_port_class

Port class of the source resource. Set to one of the following:

CALL_SW_PORT_CHANNEL_DEF	0x00
CALL_SW_PORT_BUS_DEF	0x0E
CALL_SW_PORT_NETWORK_DEF	0x0F

args.src_port_unit

Port unit number for the port class, starting from 0. If *args.src_port_class* is CALL_SW_PORT_CHANNEL_DEF, then the value indicates the total number of channels.

args.src_stream

The number of available streams for the port unit.

args.src_slot

The number of available slots for each stream.

args.port_type

A value specifying the port type of the port unit. Valid values are:

CALL_SW_PORT_CHANNEL_DEF	0x00
CALL_SW_PORT_H100_DEF	0x05
CALL_SW_PORT_T1_DEF	0x06
CALL_SW_PORT_E1_DEF	0x07
CALL_SW_PRI_T1_DEF	0x08
CALL_SW_PRI_E1_DEF	0x09
CALL_SW_AT_MODEM_DEF	0x0A
CALL_SW_PORT_BRI_DEF	0x0B
CALL_SW_PORT_ANALOG_LOOP_START_DEF	0x0C
CALL_SW_PORT_ANALOG_DID_DEF	0x0D

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Each call returns information about one connectable port unit. Call repeatedly until the return value = 0 (no more information).

For each value of *args.src_port_class* and *args.src_port_unit*, *args.src_stream* indicates the number of available streams starting from 0, and *args.src_slot* indicates the number of available slots starting from 0 for each stream.

See Also

BfvCallSWConnect, *BfvCallSWClearConns*

Example

```
BTLINE *lp;
struct args_tel_ctrl_call_sw args;

BT_ZERO(args);
while (BfvCallSWGetInfo(lp,&args) > 0)
    printf("Port class %X, Unit %d, Stream %d, Slot %d,
            Type %X\n",
        args.src_port_class,args.src_port_unit,
        args.src_stream,args.src_slot,args.port_type);
```

BfvNetworkConfigGet

Purpose

Returns network configuration information about a specified interface.

Syntax

```
int
BfvNetworkConfigGet(lp,args)
                        BTLINE                    *lp;
                        struct args_network_config*args;
```

The structure contains the following fields.

Input Fields

```
unsigned unit;
```

Output Fields

```
struct bt_sockaddr_in localAddr;
struct bt_sockaddr_in localNetmask;
struct bt_sockaddr_in gateway1;
unsigned speed;
unsigned duplex;
unsigned flowControl;
unsigned ARPTimeout;
RES *res;
```

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.unit

Unit number of the interface to query. Values are:

1 to max Ethernet port.

Output

Return value:

1	Unit information returned.
0	Unit not found.
<0	Error.

`args.localAddr`

The local address of the unit. The application fills in the *sin_family* and *sin_addr* fields of the structure type *struct bt_sockaddr_in* as follows:

```
struct bt_sockaddr_in
{
    short sin_family;
    unsigned short sin_port;
    struct bt_in_addr sin_addr;
    char sin_zero[8];
};
```

The *args.localAddr.sin_family* field specifies the address family. The only valid value is:

`TELE_CTRL_AF_INET_DEF` 0x02

The *args.localAddr.sin_addr* field specifies the network address, in network order. (This is big-endian order for `TELE_CTRL_AF_INET_DEF`).

The definition of the *struct bt_in_addr* structure is:

```
struct bt_in_addr
{
     union
     {
         struct {
             unsigned char s_b1, s_b2, s_b3, s_b4;
         } S_un_b;
         struct {
             unsigned short s_w1, s_w2;
         } S_un_w:
         unsigned long S_addr;
     } S_un;
};
```

`args.localNetmask`

The local network mask of the unit. This is the *args.localAddr.sin_addr* field of the structure *struct bt_sockaddr_in* defined in *args.localAddr*. Only the field *args.localAddr.sin_addr* is used. The address family is presumed to be the same as that specified for *args.localAddr*.

`args.gateway1`

The network address of the default gateway of the unit. This is the *args.localAddr.sin_addr* field of the structure *struct bt_sockaddr_in* defined in *args.localAddr*. Only the field *args.localAddr.sin_addr* is used. The address family is presumed to be the same as that specified for *args.localAddr*.

`args.speed`

The speed. When `TELE_CTRL_SPEED_AUTO_DEF` is selected for *args.speed*, auto-negotiation is performed for both speed and duplex (overriding the *args.duplex* selection). Valid values are:

`TELE_CTRL_SPEED_AUTO_DEF`	0x00
`TELE_CTRL_SPEED_10M_DEF`	0x01
`TELE_CTRL_SPEED_100M_DEF`	0x02

`args.duplex`

Full/half duplex control. When the application sets *args.speed* to `TELE_CTRL_SPEED_AUTO_DEF`, auto-negotiation is performed for both speed and duplex (overriding the *args.duplex* selection). Valid values are:

`TELE_CTRL_DUPLEX_HALF_DEF`	0x01
`TELE_CTRL_DUPLEX_FULL_DEF`	0x02

`args.flowControl`

Level of flow control for full duplex. Valid values are:

`TELE_CTRL_FULLDUP_AUTO_DEF`	0x00
`TELE_CTRL_FULLDUP_RECEIVE_ONLY_DEF`	0x01
`TELE_CTRL_FULLDUP_TRANSMIT_ONLY_DEF`	0x02
`TELE_CTRL_FULLDUP_BOTH_DEF`	0x03

`args.ARPTimeout`

ARP cache expiration time, in milliseconds.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Example

```
BTLINE *lp;
struct args_network_config args;

BT_ZERO(args);
args.unit = 0;
BfvNetworkConfigGet(lp,&args);
printf("Local address is: %d.%d.%d.%d\n",
    args.localAddr.sin_addr.S_un.s_b1,
    args.localAddr.sin_addr.S_un.s_b2,
    args.localAddr.sin_addr.S_un.s_b3,
    args.localAddr.sin_addr.S_un.s_b4);
...
```

BfvNetworkConfigSet

Purpose

Performs network configuration on an Ethernet interface using a configuration file and specific supplied parameters.

Syntax

```
int

BfvNetworkConfigSet(lp,args)
   BTLINE                    *lp;
   struct args_network_config*args;
```

The structure contains the following fields.

Input Fields

```
unsigned unit;
unsigned ARPflush;
unsigned set_localAddr;
struct bt_sockaddr_in localAddr;
unsigned set_localNetmask;
struct bt_sockaddr_in localNetmask;
unsigned set_gateway1;
struct bt_sockaddr_in gateway1;
unsigned set_speed;
unsigned speed;
unsigned set_duplex;
unsigned duplex;
unsigned set_flowControl;
unsigned flowControl;
unsigned set_ARPTimeout;
unsigned ARPTimeout;
```

Output Fields

```
RES *res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

`args.unit`

Unit number of the interface to query. Values are:

1 to max Ethernet port.

`args.ARPflush`

Resets the ARP cache for the unit, which is specified by *args.unit*. Not normally performed.

`args.set_localAddr`

If nonzero, sets the value of the *args.localAddr* to active.

`args.localAddr`

The local address of the unit. This is specified by filling the *sin_family* and *sin_addr* fields of the structure type *struct bt_sockaddr_in* used for this input field.

```
struct bt_sockaddr_in
{
    short sin_family;
    unsigned short sin_port;
    struct bt_in_addr sin_addr;
    char sin_zero[8];
};
```

The *args.localAddr.sin_family* field specifies the address family. The only valid value is:

`TELE_CTRL_AF_INET_DEF` 0x02

The *args.localAddr.sin_addr* field specifies the network address, in network order. (This is big-endian order for `TELE_CTRL_AF_INET_DEF`).

The definition of the *struct bt_in_addr* structure is:

```
struct bt_in_addr
{
     union
     {
         struct {
             unsigned char s_b1, s_b2, s_b3, s_b4;
         } S_un_b;
         struct {
             unsigned short s_w1, s_w2;
         } S_un_w:
         unsigned long S_addr;
     } S_un;
};
```

`args.set_localNetmask`

If nonzero, the *args.localNetmask* parameter is active.

`args.localNetmask`

The local network mask of the unit. This is the *args.localAddr.sin_addr* field of the structure *struct bt_sockaddr_in* defined in *args.localAddr*. Only the field *args.localAddr.sin_addr* is used. The address family is presumed to be the same as that specified for *args.localAddr*.

`args.set_gateway1`

If nonzero, the *args.gateway1* parameter will be active.

`args.gateway1`

The network address of the default gateway of the unit. This uses the *struct bt_sockaddr_in* structure defined in *args.localAddr*. Only the field *args.localAddr.sin_addr* is used. The address family is presumed to be the same as that specified for *args.localAddr*.

`args.set_speed`

If nonzero, the *args.speed* parameter will be active.

`args.speed`

The speed. When `TELE_CTRL_SPEED_AUTO_DEF` is selected for *args.speed*, auto-negotiation is performed for both speed and duplex (overriding the *args.duplex* selection). Valid values are:

`TELE_CTRL_SPEED_AUTO_DEF`	0x00
`TELE_CTRL_SPEED_10M_DEF`	0x01
`TELE_CTRL_SPEED_100M_DEF`	0x02

`args.set_duplex`

If nonzero, the *args.duplex* parameter will be active.

`args.duplex`

Full/half duplex control. When `TELE_CTRL_SPEED_AUTO_DEF` is selected for *args.speed*, auto-negotiation is performed for both speed and duplex (overriding the *args.duplex* selection). Valid values are:

`TELE_CTRL_DUPLEX_HALF_DEF`	0x01
`TELE_CTRL_DUPLEX_FULL_DEF`	0x02

`args.set_flowControl`

If nonzero, the *args.flowControl* parameter will be active.

`args.flowControl`

Level of flow control for full duplex. Valid values are:

`TELE_CTRL_FULLDUP_AUTO_DEF`	0x00
`TELE_CTRL_FULLDUP_RECEIVE_ONLY_DEF`	0x01
`TELE_CTRL_FULLDUP_TRANSMIT_ONLY_DEF`	0x02
`TELE_CTRL_FULLDUP_BOTH_DEF`	0x03

`args.set_ARPTimeout`

If nonzero, the *args.ARPTimeout* parameter will be active.

`args.ARPTimeout`

ARP cache expiration time, in milliseconds.

Output

Return value:

0	Success.
>0	Operational error reported by firmware.
–2	File used but module not found.
–3	File used but unit not found.
<0	Other error.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

When the `_AUTO` option is selected for *args.speed* or *args.flowControl*, the Ethernet interface negotiates the appropriate settings with link initialization.

Normally, you should allow the Ethernet interface to auto-negotiate its speed, duplex and flow control. However, the Bfv API allows you to force the speed to support various potential debugging needs. If you intend to force the interface speed (for example, to 10 Mbps or 100 Mbps), you must first allow the Ethernet interface to auto-negotiate its natural speed. The default power-up configuration is to auto-negotiate. Therefore, you can force a fixed speed the first time that you call *BfvNetworkConfigSet*. However, if you need to change the speed again, you must first call *BfvNetworkConfigSet* with the auto-negotiation setting (`TELE_CTRL_SPEED_AUTO_DEF`), wait to be sure that the auto-negotiation has had time to complete (500 ms should be adequate), and then set to the new fixed speed as desired.

Example

➢ *Setting parameters directly*

```
BTLINE *lp;
struct args_network_config args;

/* This will set the local address, leaving other
   parameters set to their previous values. */
BT_ZERO(args);
args.unit = 0;
args.set_localAddr = 1;
args.localAddr.sin_family = TELE_CTRL_AF_INET_DEF;
args.localAddr.sin_addr.S_un.s_b1 = 192;
args.localAddr.sin_addr.S_un.s_b2 = 168;
args.localAddr.sin_addr.S_un.s_b3 = 0;
args.localAddr.sin_addr.S_un.s_b4 = 1;
BfvNetworkConfigSet(lp,&args);
```

BfvNetworkQuery

Purpose

Retrieves statistics about the specified Ethernet unit.

Syntax

```
int

BfvNetworkQuery    (lp,args)
   BTLINE                    *lp;
   struct args_network_query*args;
```

The structure contains the following fields.

Input Fields

```
unsigned unit;
unsigned report_mask;
```

Output Fields

```
ENet_MIB ethernet;
ARP_MIB arp;
UDP_MIB udp;
IF_MIB if;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.unit

Ethernet unit for generating the report will be generated.

args.report_mask

Specifies values to use when generating reports. The value is formed by ORing together the following values:

TELE_CTRL_REPORT_ENET_DEF	0x01
TELE_CTRL_REPORT_ARP_DEF	0x10

TELE_CTRL_REPORT_UDP_DEF	0x20
TELE_CTRL_REPORT_IF_DEF	0x40

Output

Return value:

0	Success.
<0	Error.

args.if

A structure of IF MIB compatible fields, filled in when *args.report_mask* contains TELE_CTRL_REPORT_UDP_DEF. Refer to *Table 4*, IF Statistics in the details section.

args.ethernet

A structure of Ethernet MIB compatible fields, filled in when *args.report_mask* contains TELE_CTRL_REPORT_ENET_DEF. Refer to *Table 5*, Ethernet Statistics in the details section.

args.arp

A structure of up to 60 ARP MIB compatible fields, filled in when *args.report_mask* contains TELE_CTRL_REPORT_ARP_DEF. Refer to *Table 6*, ARP Statistics in the details section.

args.udp

A structure of UDP MIB compatible fields, filled in when *args.report_mask* contains TELE_CTRL_REPORT_UDP_DEF. Refer to *Table 7*, UDP Statistics in the details section.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Bits set in *args.report_mask* determine which output values will be returned.

The output structures are defined in the header file *btmib.h*. Not all statistics are available if the unit is part of a trunk.

Table 4. IF Statistics

ifHighSpeed	Indicates 10/100Mbps. If auto-negotiation is enabled, but operational speed has not been negotiated, maximum speed supported by the interface shall be reported (100Mbps).
ifHCInOctets	Number of octets in valid MAC frames received on this interface, including the MAC header and FCS.
ifHCInErrors	Sum of receive errors: 40 - Receive CRC Errors 44 - Receive Alignment Errors 48 - Receive Resource Errors 52 - Receive Overrun Errors 60 - Receive Short Frame Errors
ifInUnknownProtos	For packet-oriented interfaces, the number of packets received via the interface which were discarded because of an unknown or unsupported protocol.
ifHCOutOctets	The total number of octets transmitted out of the interface, including framing characters.
ifInUcastPkts	The total number of packets received which were not addressed to a multicast or broadcast address at this sub-layer, including those that were discarded or not sent.
ifInMulticastPkts	The number of packets received, which were addressed to a multicast address at this sub-layer.
ifInBroadcast	The number of packets received, which were addressed to a broadcast address.
ifOutUcastPkts	The total number of packets that higher-level protocols requested be transmitted, and which were not addressed to a multicast or broadcast address at this sub-layer, including those that were discarded or not sent.
ifOutMulticastPkts	The number of packets sent, which were addressed to a multicast address.
ifOutBroadcastPkts	The number of packets sent, which were addressed to a broadcast address.
ifPhysAddress	Hardware MAC address

Table 5. Ethernet Statistics

dot3StatsExcessiveCollisions	The number of frames for which transmission on a particular interface fails due to excessive collisions.
dot3StatsLateCollisions	The number of times that a collision is detected on a particular interface later than 512 bit-times into the transmission of a packet.
dot3StatsInternalMacTransmitErrors	The number of frames for which transmission on a particular interface fails due to an internal MAC sublayer transmit error.
dot3StatsCarrierSenseErrors	The number of times that the carrier sense condition was lost or never asserted when attempting to transmit a frame on a particular interface.
dot3StatsDeferredTransmissions	The number of frames for which the first transmission attempt on a particular interface is delayed because the medium is busy.
dot3StatsSingleCollisionFrames	The number of successfully transmitted frames on a particular interface for which transmission is inhibited by exactly one collision.
dot3StatsFCSErrors	The number of frames received on a particular interface that are an integral number of octets in length but do not pass the FCS check.
dot3StatsAlignmentErrors	The number of frames received on a particular interface that are not an integral number of octets in length and do not pass the FCS check.
dot3StatsInternalMacReceiveErrors	The number of frames for which reception on a particular interface fails due to an internal MAC sublayer receive error.
dot3StatsXTransmitGoodFrames	The number of frames transmitted successfully by this device.
dot3StatsXTransmitFlowControls	The number of flow control messages transmitted by this device.
dot3StatsXReceiveGoodFrames	The number of frames successfully received without data errors by this device.
dot3StatsXReceiveResourceErrors	The number of frames received while the device's receive unit was not in the ready state.
dot3StatsXReceiveFlowControls	The number of flow control frames received.

Table 6. ARP Statistics

atPhysAddress	Hardware MAC address.
atNetAddress	IP address.

Table 7. UDP Statistics

udpInDatagrams	Total delivered datagrams.
udpNoPorts	Undelivered datagrams: unused port.
udpInErrors	Undelivered datagrams: other reasons.
udpOutDatagrams	Successfully sent datagrams.

Example

```
struct args_network_query args;

/* Request reports on Ethernet and IP. */
BT_ZERO(args);
args.unit = 0;
args.report_mask = TELE_CTRL_REPORT_ENET_DEF |
    TELE_CTRL_REPORT_IP_DEF;
BfvNetworkQuery(lp, &args);
```

BfvTelephGetInfo

Purpose

Retrieves and returns information about the telephony hardware units on the current module and their available port types.

Syntax

```
int

BfvTelephGetInfo  (lp, args)
   BTLINE                  *lp;
   struct args_tel_ctrl_call_sw*args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

```
unsigned src_port_unit;
unsigned port_type;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value:

1	Unit information is returned.
0	No more unit information.
<0	Error.

args.src_port_unit

Port unit number, starting from 0.

args.port_type

A value defining available port types for the current telephony hardware unit. The value is formed by logically ORing together the following values:

TELE_CTRL_PORT_ANALOG_LOOP_START_DEF	0x01
TELE_CTRL_PORT_ANALOG_DID_DEF	0x02
TELE_CTRL_PORT_H100_DEF	0x20
TELE_CTRL_PORT_T1_DEF	0x40
TELE_CTRL_PORT_E1_DEF	0x80
TELE_CTRL_PORT_PRI_T1_DEF	0x100
TELE_CTRL_PORT_PRI_E1_DEF	0x200
TELE_CTRL_PORT_AT_MODEM_DEF	0x400
TELE_CTRL_PORT_BRI_DEF	0x800
TELE_CTRL_PORT_TYPE_H100_SCBUS_DEF	0x4000
TELE_CTRL_PORT_TYPE_H100_MVIP_DEF	0x8000
TELE_CTRL_PORT_TYPE_T3_DEF	0x10000

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

Each call returns information about one hardware unit. Call repeatedly until the return value = 0 (no more information).

See Also

BfvCallSWConnect, *BfvCallSWGetInfo*

Example

```
BTLINE *lp;
struct args_tel_ctrl_call_sw args;

BT_ZERO(args);
while (BfvTelephGetInfo(lp,&args) > 0)
    printf("Unit %d, port type %X\n",
        args.src_port_unit,args.port_type);
```

BfvTelephReset

Purpose

Modifies the telephony state on the current module to permit performing high level configuration again using *BfvCallCtrlInit*.

Syntax

```
void
BfvTelephReset     (lp, args)
   BTLINE                    *lp;
   struct args_tel_ctrl_call_sw*args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

This function only resets the configuration of the module when the application calls the *BfvCallCtrlInit* function to perform the modification. If not called, *BfvCallCtrlInit* will not do any configuration after the first time.

See Also

BfvCallSWConnect, *BfvCallCtrlInit*, *BfvTelephGetInfo*

Example

See the *telreset.c* application in the *bapp.src* directory.

BfvTelephSave

Purpose

Saves already configured telephony parameters to non-volatile memory (NVRAM) on the current module.

Syntax

```
int

BfvTelephSave      (lp, args)
   BTLINE                  *lp;
   struct args_tel_ctrl_call_sw*args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value:

0	Success
>0	Operational error reported by firmware
<0	Other error

TELE_CTRL_NVRAM_READ_SSEEPROM_ERR_DEF	0x02
TELE_CTRL_NVRAM_WRITE_SSEEPROM_ERR_DEF	0x03
TELE_CTRL_NVRAM_SSEEPROM_WRITTEN_DEF	0x04

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The process this function performs enables boards to power up and initialize with legal configurations for their environment.

See Also

`BfvCallCtrlInit`

Example

See the *telsave.c* sample application in the *bapp.src* directory.

5 - Status and Monitoring

This chapter describes functions to control and monitor the status of modules and telephony or network ports.

With the status and monitoring functions, you can:

- Set and get the state of a module by reading the Board Status LED.
- Set a module temperature threshold.
- Get the temperature of a module.
- Have a module perform a series of self tests and, optionally, report the results.
- Have a module notify the application of events or conditions on the module such as:
  - A network alarm
  - A network error
  - An H.110 clock event
  - A temperature alarm
  - General status of the module
  - Ethernet link status
  - RTP/RTCP statistics and reports

Function Summary

Table 8 provides a brief summary of the functions used for monitoring and retrieving the status of a board.

Table 8. Board Status and Monitoring Function Summary

Function	Purpose	Page
BfvBoardNotify	Enables or disables module level notification and sets up an optional callback function.	*171*
BfvBoardStateGet	Retrieves the current state of a module (Board Status LED).	*178*
BfvBoardStateSet	Sets the current state of a module (Board Status LED).	*180*
BfvBoardTemperatureGet	Retrieves the current temperature of a module.	*182*
BfvBoardTemperatureThreshSet	Sets the temperature threshold of a module for the purpose of module event notification.	*184*
BfvBoardTest	Initiates a series of self tests on the module and reports the results as they occur through an optional callback function.	*186*
BfvIPCallControlNotify	Enables or disables IP Call Control module notifications and sets up an optional callback function.	*190*
BfvRtpEventControl	Registers for RTP events, statistics, and RTCP reports.	*195*
BfvRtpEventGet	Retrieves RTP events, statistics, and RTCP reports.	*198*
BfvRtcpReportSend	Configures RTCP SDES items and generates RTCP APP reports.	*206*

BfvBoardNotify

Purpose

Enables or disables module level notification and sets up an optional callback function.

Syntax

```
void
BfvBoardNotify                (lp, args)
   BTLINE                     *lp;
   struct args_board_notify*args;
```

The structure contains the following fields.

Input Fields

```
unsigned enable;
unsigned notify_type;
void (*func)(BTLINE *lp, unsigned notify_type,
    struct args_board_notify *args);
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.enable

0	Disables notification.
Nonzero	Enables notification.

This applies only to the notification type indicated by *args.notify_type*.

`args.notify_type`

The notification type to enable or disable.

Valid values are:

`ADMIN_NOTIFY_NETWORK_ALARM_DEF`	0x01
`ADMIN_NOTIFY_NETWORK_ERROR_DEF`	0x02
`ADMIN_NOTIFY_H110_CLOCK_EVENT_DEF`	0x03
`ADMIN_NOTIFY_TEMP_ALARM_DEF`	0x04
`ADMIN_NOTIFY_BOARD_STATE_DEF`	0x05
`ADMIN_NOTIFY_ENET_LINK_DEF`	0x06

`args.func`

Optional callback function. If `NULL`, no callback will be performed. The function will be called as indicated by its definition above. *Notify_type* contains the notification type that occurred. *Args* is a pointer to an *args_board_notify* structure with fields set to values relevant to the incoming notification.

Note: The structure and its contents must not be modified.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

For the current set of notification types, the list below indicates the fields of the *args_board_notify* structure that will be used when the callback function is called. The text that follows provides values and definitions for these fields.

`ADMIN_NOTIFY_NETWORK_ALARM_DEF`

unit, network_alarm

`ADMIN_NOTIFY_NETWORK_ERROR_DEF`

unit, network_error

`ADMIN_NOTIFY_H110_CLOCK_EVENT_DEF`

h110_clock_status

`ADMIN_NOTIFY_TEMP_ALARM_DEF`

temp_alarm, temp

`ADMIN_NOTIFY_BOARD_STATE_DEF`
board_state

`ADMIN_NOTIFY_ENET_LINK_DEF`
unit, link_status, failover_unit, speed, duplex

`args->unit`

Telephony or network hardware unit for notification.

`args->network_alarm`

Network alarm type. Valid values are:

`ADMIN_UNIT_ALARM_NONE_DEF`	0x01
`ADMIN_UNIT_ALARM_LOS_DEF`	0x02
`ADMIN_UNIT_ALARM_RAI_DEF`	0x03
`ADMIN_UNIT_ALARM_AIS_DEF`	0x04

`args->network_error`

Network error type. Valid values are:

`ADMIN_UNIT_ERROR_NONE_DEF`	0x01
`ADMIN_UNIT_ERROR_FRAMING_DEF`	0x02
`ADMIN_UNIT_ERROR_CRC_DEF`	0x03
`ADMIN_UNIT_ERROR_BPV_DEF`	0x04
`ADMIN_UNIT_ERROR_SLIP_DEF`	0x05

`args->h110_clock_status`

H.110 clock status. Valid values are:

`ADMIN_H110_MASTER_REF_A_DEF` 0x07

Clock is from first T1/E1 port.

`ADMIN_H110_MASTER_REF_B_DEF` 0x08

Clock is from second T1/E1 port.

`ADMIN_H110_MASTER_REF_NETREF1_DEF` 0x09

Clock is from NETREF1.[a]

`ADMIN_H110_MASTER_REF_NETREF2_DEF` 0x0A

Clock is from NETREF2.[a]

`ADMIN_H110_MASTER_REF_INTERNAL_DEF` 0x0B

Clock is from internal oscillator.

`ADMIN_H110_MASTER_REF_C_DEF` 0x0C

Clock is from third T1/E1 port.

`ADMIN_H110_MASTER_REF_D_DEF` 0x0D

Clock is from fourth T1/E1 port.

`ADMIN_H110_MASTER_REF_E_DEF` 0x0E

Clock is from fifth T1/E1 port.

`ADMIN_H110_MASTER_REF_F_DEF` 0x0F

Clock is from sixth T1/E1 port.

`ADMIN_H110_MASTER_REF_G_DEF` 0x10

Clock is from seventh T1/E1 port.

`ADMIN_H110_MASTER_REF_H_DEF` 0x11

Clock is from eighth T1/E1 port.

`ADMIN_H110_MASTER_REF_A_BUS_DEF` 0x12

Clock is from H110 A bus.

`ADMIN_H110_MASTER_REF_B_BUS_DEF` 0x13

Clock is from H110 B bus.

`ADMIN_H110_MASTER_REF_NONE_DEF` 0x14

Not receiving valid clock signal.

[a.] See the Release Notes to determine if supported.

`args->temp_alarm`

Temperature alarm threshold in units of 1/2 degree Celsius.

`args->temp`

Temperature of the module in units of 1/2 degree Celsius. An alarm event triggers when the module's temperature first exceeds the threshold or first becomes less than the threshold. In other words, an alarm is generated once for each threshold crossing, regardless of the direction.

`args->board_state`

Module State. Valid values are:

`ADMIN_BOARD_STATE_OK_DEF`	0x01
`ADMIN_BOARD_STATE_WARNING_DEF`	0x02
`ADMIN_BOARD_STATE_ERROR_DEF`	0x03
`ADMIN_BOARD_STATE_OUTOFSERVICE_DEF`	0x04
`ADMIN_BOARD_STATE_OFFLINE_DEF`	0x05

`args->link_status`

Ethernet link status. Valid values are:

`ADMIN_ENET_LINK_DOWN_DEF`

The link is down.

`ADMIN_ENET_LINK_UP_DEF`

The link is up.

`ADMIN_ENET_LINK_FAILOVER_DEF`

The link has failed over to another unit.

`ADMIN_ENET_LINK_FAILOVER_TRUNK_DEF`

The link has failed over to another unit within the trunk.

`ADMIN_ENET_LINK_FAILOVER_UNSUCC_DEF`

The link attempted to failover but was unsuccessful.

`args->failover_unit`

Ethernet failover unit. Valid only when *`args->link_status`* is `ADMIN_ENET_LINK_FAILOVER_DEF`.

`args->speed`

Ethernet link speed. Valid only when *`args->link_status`* is ADMIN_ENET_LINK_UP_DEF. Valid values are:

`ADMIN_SPEED_AUTO_DEF`	0x0
`ADMIN_SPEED_10M_DEF`	0x1
`ADMIN_SPEED_100M_DEF`	0x2

`args->duplex`

Ethernet link duplex mode. Valid only when *args->link_status* is `ADMIN_ENET_LINK_UP_DEF`. Valid values are:

`ADMIN_DUPLEX_AUTO_DEF`	0x0
`ADMIN_DUPLEX_HALF_DEF`	0x1
`ADMIN_DUPLEX_FULL_DEF`	0x2

Enabling notification with no callback function is useful for the purpose of viewing the resulting notifications in Bfv API debug mode.

Only one notification function can be set up per line pointer, though the notifications for the different notification types are set up separately. All calls made with a non-NULL *args.func* value should contain the same value.

The callback function will be called when notification is first enabled and when a new notification occurs. It might, however, sometimes be called multiple times in succession with the same *args->notify_type* value under some conditions. This is normal and does not indicate a recurrence of the event causing the notification. Applications should only interpret the notification as representing an event of some sort when the *args->notify_type* value or associated *args* field values are different from the previous value.

The *args_board_notify* structure passed to the callback function also contains an *args_packet* structure pointer. This pointer can be used to access tagged values of a notification event in case a new notification type is implemented which the Bfv API does not yet support. Such access requires use of low level command set functions and specific knowledge of the command set and should only be done under the guidance of Dialogic Technical Services and Support.

See Also

BfvBoardTemperatureThreshSet, *BfvBoardStateGet*, *BfvBoardStateSet*

Example

```
/* Set up function to receive network alarm notifications.
   */

void notify_func(lp,notify_type,args)
BTLINE *lp;
unsigned notify_type;
struct args_board_notify args;
{
    printf("Got notify for type %d\n",notify_type);
    if (notify_type == ADMIN_NOTIFY_NETWORK_ALARM_DEF)
        printf("Network alarm type %x on unit %d\n",
        args->network_alarm,args->unit);
}

main()
{
    BTLINE *lp;
    struct args_board_notify args;

    ...
    BT_ZERO(args);
    args.enable = 1;
    args.notify_type = ADMIN_NOTIFY_NETWORK_ALARM_DEF;
    args.func = notify_func;
    BfvBoardNotify(lp,&args);

    /* Call BfvRcvProcessPkt or other API calls here. */
}
```

BfvBoardStateGet

Purpose

Retrieves the current state of a module.

Syntax

```
void
BfvBoardStateGet            (lp, args)
   BTLINE                   *lp;
   struct args_board_state  *args;
```

The structure contains the following fields.

Output Fields

```
unsigned state;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.state

Module state. Call *BfvBoardStateSet* before using *BfvBoardStateGet*. The valid values are:

ADMIN_BOARD_STATE_UNINITIALIZED_DEF	0x00
ADMIN_BOARD_STATE_OK_DEF	0x01
ADMIN_BOARD_STATE_WARNING_DEF	0x02
ADMIN_BOARD_STATE_ERROR_DEF	0x03
ADMIN_BOARD_STATE_OUTOFSERVICE_DEF	0x04
ADMIN_BOARD_STATE_OFFLINE_DEF	0x05

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

The SR140 does not support this function.

Each Dialogic® Brooktrout® module has a Board Status LED that indicates a module state previously set by using the *BfvBoardStateSet* function. Use the *BfvBoardStateGet* function to retrieve the current state. The valid states are:

LED States	Flashing Patterns	
`ADMIN_BOARD_STATE_UNINITIALIZED_DEF`		
Indicates that *BfvBoardStateSet* has not been used to set a state.		
`ADMIN_BOARD_STATE_OK_DEF`	1/4 sec. on 1/2 sec. off	Green
`ADMIN_BOARD_STATE_WARNING_DEF`	1/4 sec. on 1/2 sec. off	Yellow
`ADMIN_BOARD_STATE_ERROR_DEF`	constant on	Red
`ADMIN_BOARD_STATE_OUTOFSERVICE_DEF`	1/4 sec. on 1/2 sec. off	Red
`ADMIN_BOARD_STATE_OFFLINE_DEF`	constant off	–

See Also

BfvBoardStateSet

Example

```
BTLINE *lp;
struct args_board_state args;

...
BT_ZERO(args);
BfvBoardStateGet(lp,&args);
if (args.res.status == BT_STATUS_OK)
    printf("Module state is %d\n",args.state);
```

BfvBoardStateSet

Purpose

Sets the module state value.

Syntax

```
void
BfvBoardStateSet            (lp, args)
   BTLINE                   *lp;
   struct args_board_state  *args;
```

The structure contains the following fields.

Input Fields

```
unsigned state;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.state

New module state. Call *BfvBoardStateSet* before using *BfvBoardStateGet*.

The valid values are:

ADMIN_BOARD_STATE_OK_DEF	0x01
ADMIN_BOARD_STATE_WARNING_DEF	0x02
ADMIN_BOARD_STATE_ERROR_DEF	0x03
ADMIN_BOARD_STATE_OUTOFSERVICE_DEF	0x04
ADMIN_BOARD_STATE_OFFLINE_DEF	0x05

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

The SR140 does not support this function.

This function allows you to set the state of the Board Status LED indicator on the module. It does not affect the operation of any other Bfv API function nor alter the behavior of the module, other than changing the color or winking pattern of the LED. The valid states are:

LED States	Flashing Patterns	
`ADMIN_BOARD_STATE_OK_DEF`	1/4 sec. on 1/2 sec. off	Green
`ADMIN_BOARD_STATE_WARNING_DEF`	1/4 sec. on 1/2 sec. off	Yellow
`ADMIN_BOARD_STATE_ERROR_DEF`	constant on	Red
`ADMIN_BOARD_STATE_OUTOFSERVICE_DEF`	1/4 sec. on 1/2 sec. off	Red
`ADMIN_BOARD_STATE_OFFLINE_DEF`	constant off	–

See Also

BfvBoardStateGet

Example

```
BTLINE *lp;
struct args_board_state args;

...
/* Set state to Warning */
BT_ZERO(args);
args.state = ADMIN_BOARD_STATE_WARNING_DEF;
BfvBoardStateSet(lp,&args);
```

BfvBoardTemperatureGet

Purpose

Retrieves the current temperature of a module.

Syntax

```
void
BfvBoardTemperatureGet      (lp, args)
   BTLINE                   *lp;
   struct args_temperature  *args;
```

The structure contains the following fields.

Output Fields

```
int current_temp;
RES res;
```

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.current_temp

Temperature, in 1/2 degree Celsius units.

1000 = Invalid. The module does not support this function.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function does not support the SR140 and the following boards because they do not have a temperature sensor:

- TR1034 Analog and BRI boards
- TruFax® PCI BRI boards

See Also

BfvBoardTemperatureThreshSet

Example

```
struct args_temperature args;

BT_ZERO(args);
BfvBoardTemperatureGet(lp,&args);
printf("Temp is %d in 0.5 degree
        units\n",args.current_temp);
```

BfvBoardTemperatureThreshSet

Purpose

Sets the temperature threshold of a module for the purpose of module event notification.

Syntax

```
void
BfvBoardTemperatureThreshSet(lp, args)
   BTLINE                    *lp;
   struct args_temperature *args;
```

The structure contains the following fields.

Input Fields

```
int temp_thresh;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.temp_thresh

Temperature threshold, in 1/2 degree Celsius units.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

See Also

BfvBoardTemperatureGet, *BfvBoardNotify*

Example

```
struct args_temperature args;

/* Set temp thresh to 80 Celsius */
BT_ZERO(args);
args.temp_thresh = 160;
BfvBoardTemperatureThreshSet(lp,&args);
```

BfvBoardTest

Purpose

Initiates a series of self tests of the module and reports the results as they occur through an optional callback function.

Syntax

```
void
BfvBoardTest                (lp, args)
   BTLINE                   *lp;
   struct args_board_test   *args;
```

The structure contains the following fields.

Input Fields

```
unsigned iterations;
unsigned test_mask;
void (*func)(BTLINE *lp, struct args_board_test
             *args);
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.iterations

The number of test iterations to perform.

`args.test_mask`

An indication of which tests to perform. Logically OR together one or more of the following values:

`ADMIN_BOARD_TEST_DRAM_DEF` 0x01

DRAM (dynamic memory)

`ADMIN_BOARD_TEST_HPI_DEF` 0x02

Communication path to DSPs

`ADMIN_BOARD_TEST_NVRAM_DEF` 0x04

NVRAM (nonvolatile memory)

`ADMIN_BOARD_TEST_GUPI_DEF` 0x08

Programmable logic controlling processor interfaces

`ADMIN_BOARD_TEST_PCM_CONTROLLER_DEF` 0x10

PCM controller

`ADMIN_BOARD_TEST_SLBSRAM_USING_PPC_DEF` 0x40

Shared local bus SRAM (static RAM) access from control processor

`ADMIN_BOARD_TEST_SLBSRAM_USING_DSP_DEF` 0x80

Shared local bus SRAM (static RAM) access from DSP

`ADMIN_BOARD_TEST_ENET_CONTROLLER_DEF` 0x100

Ethernet controller

Note: Some of these tests might not be supported in a particular version. For more information, see the Release Notes that apply to your Dialogic® Brooktrout® board type.

`args.func`

Optional callback function. If `NULL`, no callback will be performed.

The function will be called as indicated by its definition above. The *args* variable is a pointer to an *args_board_test* structure with fields set to values indicating the current test status. The structure and its contents must not be modified.

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 does not support this function.

BfvBoardTest does not require that the firmware be loaded to the module to function. Also, since the *BfvBoardTest* function performs a module reset, firmware will need to be downloaded to the module after *BfvBoardTest* executes and before any other function is attempted (even if firmware had previously been downloaded to the module).

Note: Use this function only with TR1034 digital (T1/E1) boards.

Each time the callback function is called, the following fields are used:

args->test_mask	=	Indication of which test has started or completed.
args->status	=	Whether test started or completed/failed.
args->err_info	=	Additional error information.

Calling this function will cause the module to be reset back to its initial state. After the test, download firmware before proceeding.

When the tests are first started, the callback function will be called with *args->test_mask* set to `MILL_TEST_ALL_DEF` (0xffffffff) and *args->status* set to `ADMIN_BOARD_TEST_STARTED_DEF`. As each test starts, the function will be called with *args->test_mask* indicating the test type and *args->status* set to `ADMIN_BOARD_TEST_STARTED_DEF`. As each test finishes, the function will be called with *args->test_mask* indicating the test type and *args->status* set to `ADMIN_BOARD_TEST_SUCCESS_DEF` or `ADMIN_BOARD_TEST_FAILED_DEF`. At the end of the entire set of all iterations of tests, the function will be called with *args->test_mask* set to `MILL_TEST_ALL_DEF` (0xffffffff) and *args->status* set to `ADMIN_BOARD_TEST_SUCCESS_DEF`.

When *args->status* is ADMIN_BOARD_TEST_FAILED_DEF, *args->error_info* might contain additional error information, depending upon the test. For ADMIN_BOARD_TEST_ENET_CONTROLLER_DEF, the least significant byte contains the Ethernet controller unit number that had an error. For ADMIN_BOARD_TEST_SLBSRAM_USING_DSP_DEF, the least significant byte contains the DSP number that had an error.

Performing tests with no callback function is useful for the purpose of viewing the results in Bfv API debug mode.

Test durations depend upon the Dialogic® Brooktrout® board type, and the *args->test_mask* in use. A typical test might take 4 seconds for all tests, excluding the DRAM test (ADMIN_BOARD_TEST_DRAM_DEF). A typical DRAM test may take 15 seconds.

Example

```
void test_func(lp,args)
BTLINE *lp;
struct args_board_test *args;
{
    printf("Test %x, status %d\n",args->test_mask,
      args->status);
}

main()
{
    BTLINE *lp;
    struct args_board_test args;

    ...
    /* Do 10 iterations of the HPI test */
    BT_ZERO(args);
    args.iterations = 10;
    args.test_mask = ADMIN_BOARD_TEST_HPI_DEF;
    args.func = test_func;
    BfvBoardTest(lp,&args);
}
```

BfvIPCallControlNotify

Purpose

Enables or disables IP Call Control module notifications and sets up an optional callback function.

Syntax

```
int
BfvIPCallControlNotify      (lp, args)
   BTLINE                   *lp;
   struct args_ipcallcontrol_notify*args;
```

The structure contains the following fields.

Input Fields

```
unsigned enable;
unsigned notify_type;
void (*func)(BTLINE *lp, unsigned notify_type,
    struct args_ipcallcontrol_notify *args);
```

Output Fields

```
RES res;
GWStatus gateway_status[BT_MAX_GATEWAYS]
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.enable

0	Disables notification.
Nonzero	Enables notification.

This applies only to the notification type indicated by *args.notify_type*.

`args.notify_type`

The notification type to enable or disable.

Valid values are:

`IP_CC_NOTIFY_GATEWAY_STATUS_DEF`	0x01

`args.func`

Optional callback function. If `NULL`, no callback will be performed. The function will be called as indicated by its definition above. *Notify_type* contains the notification type that occurred. *Args* is a pointer to an *args_ipcallcontrol_notify* structure with fields set to values relevant to the incoming notification.

Note: The structure and its contents must not be modified.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The line structure passed into this function should be allocated using BfvSessionAttach() with the HOST_MODULE() macro used to specify the value for the *mm_bModule* field in the destination address and the *mm_bChannel* field set to 1 (the Administrative channel). The value specified in the HOST_MODULE() macro should be the Host Module index of the IP Call Control stack as specified in the CALLCTRL.CFG configuration file. For example, if the CALLCTRL.CFG configuration file contains the following:

```
[host_module.1]

   module_library=brktsip.dll
```

then the HOST_MODULE() macro would be specified with a 1 for the SIP IP Call Control host module:

```
args.dest_addr.mm_bModule  = HOST_MODULE(1);
args.dest_addr.mm_bChannel = 1;
```

For the current set of notification types, the list below indicates the fields of the *args_ipcallcontrol_notify* structure that will be used when the callback function is called. The text that follows provides values and definitions for these fields.

`IP_CC_NOTIFY_GATEWAY_STATUS_DEF`

SIP Gateway status change notification

A maximum of BT_MAX_GATEWAYS (4) gateways are supported for SIP Gateway status notifications. When a notification of IP_CC_NOTIFY_GATEWAY_STATUS_DEF is received, the *GWStatus* structures pointed to by *args->gateway_status* will be populated with the current status of all of the SIP Gateways configured in the CALLCTRL.CFG configuration file as indicated in the table below:

CALLCTRL.CFG Parameter	Gateway Status Array Index
sip_default_gateway	*args->gateway_status[0]*
sip_gateway2	*args->gateway_status[1]*
sip_gateway3	*args->gateway_status[2]*
sip_gateway4	*args->gateway_status[3]*

The *GWStatus* data structure used to report the current status of the SIP Gateways is defined as follows:

```
typedef struct {
   unsigned status;
   unsigned response;
} GWStatus;
```

If a SIP Gateway isn't configured in the CALLCTRL.CFG configuration file, the *GWStatus* data structure associated with it will be populated with a *status* of IP_CC_GATEWAY_STATUS_DOWN_DEF (0) and a *response* of 0.

args->gateway_status[xx].status

SIP Gateway Status. The values returned in this field represent the current SIP Gateway status as determined by responses received for SIP OPTIONS requests transmitted to each of the SIP Gateways configured in the CALLCTRL.CFG configuration file.

Valid values are:

IP_CC_GATEWAY_STATUS_DOWN_DEF	0x00
IP_CC_GATEWAY_STATUS_UP_DEF	0x01

`args->gateway_status[xx].response`

SIP Gateway Response. The values returned in this field represent the most recent SIP responses received from SIP OPTIONS requests transmitted to the SIP Gateways configured in the CALLCTRL.CFG configuration file. A value of zero (0) indicates a timeout occurred waiting for a response.

Only one notification function can be set up per line pointer. The callback function will be called when a notification is first enabled and whenever a new notification occurs.

See Also

BfvSessionAttach

Example

```
void notify_func(BTLINE *lp,
                 unsigned notify_type,
                 struct args_ipcallcontrol_notify *args)
{
   printf("Got notify for type %d\n", notify_type);
   if (notify_type == IP_CC_NOTIFY_GATEWAY_STATUS_DEF)
   {
      int gw = 0;

      for (gw=0; gw<BT_MAX_GATEWAYS; gw++)
      {
         printf("Gateway [%d] Status: %d, Response: %d\n",
                gw,
                args->gateway_status[gw].status,
                args->gateway_status[gw].response);
      }
   }
}

main()
{
   BTLINE *lp;
   struct args_line_admin admin_args;
   struct args_ipcallcontrol_notify notify_args;

   /* Attach to SIP IP Call Control Host Module              */
   /* Host Module is the value specified in callctrl.cfg     */
   /* [host_module.1]                                        */
   /*   module_library=brktsip.dll                           */

   BT_ZERO(admin_args);
   admin_args.dest_addr.mm_bModule = HOST_MODULE(1);
   admin_args.dest_addr.mm_bChannel = 1;
   admin_args.present = 1;
   admin_args.unique  = 0;
   if ((lp = BfvSessionAttach(&admin_args)) == NULL)
   {
      printf("BfvSessionAttach failed.\n");
      /* Process attach error */
      ...
   }

   BT_ZERO(notify_args);
   notify_args.enable = 1;
   notify_args.notify_type = IP_CC_NOTIFY_GATEWAY_STATUS_DEF;
   notify_args.func = notify_func;
   BfvIPCallControlNotify(lp, &notify_args);

   /* Call BfvRcvProcessPkt or other API calls here. */
   ...
}
```

BfvRtpEventControl

Purpose

Registers for RTP events, statistics, and RTCP reports.

Syntax

```
int
BfvRtpEventControl          (lp, args)
   BTLINE                   *lp;
struct args_rtp_event_control*args;
```

The structure contains the following fields.

Input Fields

```
BTLINE *dest_lp;
int event_enable;
rtp_event_type_t event_type;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to the *args_rtp_event_control* structure containing input and output fields.

args.dest_lp

Set to NULL for the event to be returned to the BTLINE line pointer (*lp*) used to invoke *BfvRtpEventControl*. Otherwise, the event is sent to this line pointer. The same *dest_lp* may be used for more than a single module.

args.event_enable

Boolean indicating whether one or more events are enabled or disabled.

0 = Event disabled 1 = Event enabled

args.event_info.event_type

The registered events. These events can be OR'ed together.

Event	Description
RTP_EVENT_JITTER_FRAME_OVERDUE	Indicates a frame not received in time.
RTP_EVENT_JITTER_FRAME_INCORRECT_ORDER	A frame received not in correct order. Only sent if reordering is turned off.
RTP_EVENT_JITTER_OVERFLOW	Jitter buffer not dequeuing packets fast enough because, presumably, the remote side is sending too fast.
RTP_EVENT_RTCP_SRRR	Sender Report (SR) and Receiver Report (RR) RTCP packets.
RTP_EVENT_RTCP_SDES	Source description item (SDES) packets.
RTP_EVENT_RTCP_BYE	Disconnection (BYE) RTCP packets.
RTP_EVENT_RTCP_APP	Application specific RTCP packets.
RTP_EVENT_RTCP_ALL	Only valid as report type. Reports all RTCP events.
RTP_EVENT_RTP_PAYLOAD_CHANGE	An RTP packet with a payload different than the one negotiated has been received and ignored.
RTP_EVENT_PACKET_STATS	RTP statistics. Enabling this event will generate an immediate event from the RTP stack.
RTP_EVENT_ALL	Selects all possible events.

Output

Return values:

0 = The function executed successfully
<0 = An error occurred

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

*BfvRtpEventContro*l registers for RTP events, statistics, and RTCP reports. The events are retrieved by the application using *BfvRtpEventGet*.

Some RTCP event types, such as `RTP_EVENT_RTCP_ALL` and `RTP_EVENT_RTCP_SDES`, can result in multiple returns from *BfvRtpEventGet*. For example, a single RTCP report can contain a sender report (SR) and an SDES item. Both of these results return from *BfvRtpEventGet*. Enabling the `RTP_EVENT_PACKET_STATS` event generates RTP statistics immediately. If enabled events are to be disabled, they must be disabled explicitly using this function.

All events are disabled by default.

BfvRtpEventGet

Purpose

Retrieves RTP events, statistics, and RTCP reports previously registered by *BfvRtpEventControl*.

Syntax

```
int
BfvRtpEventGet              (lp, args)
   BTLINE                   *lp;
   struct args_rtp_event_control*args;
```

The structure contains the following fields.

Input Fields

```
LONG timeout;
```

Output Fields

```
int event_module_num;
int event_chan_num;
struct rtp_event
{
    rtp_event_type_t event_type;
    union
    {
        struct rtp_rtcp
        {
            int version;
            int pad_flag;
            int report_count;
            int packet_length;
            rtp_rtcp_type_t packet_type;
```

```
                union
                {
                   struct sr
                   {
                      unsigned ssrc;
                      unsigned ntp_seconds;
                      unsigned ntp_fraction;
                      unsigned rtp_timestamp;
                      unsigned num_packets_sent
                      unsigned num_bytes_sent;
                      rtp_rtcp_rr_t sr_item[31];
                   } u_sr;
                   struct rr
                   {
                      unsigned ssrc;
                      rtp_rtcp_rr_t rr_item[31];
                   } u_rr;
                   struct sdes
                   {
                      rtp_rtcp_sdes_chunk sdes_chunk[31];
                   } u_sdes;
                   struct bye
                   {
                       unsigned ssrc[31];
                       short reason_length;
                       char  reason[RTP_MAX_RTCP_\
                                          BYE_REASON];
                   } u_bye;
                   struct app
                   {
                      char app_data[RTP_MAX_RTCP_APP_DATA];
                      int app_data_len;
                   } u_app;
                } u_reports;
             } rtcp_data;
             struct rtp_packet_stats
             {
                   unsigned num_packets;
                   unsigned num_octets;
                   unsigned num_packets_lost;
                   unsigned interarrival_jitter;
                   unsigned avg_transmission_delay;
             } packet_stats;
          }u;
```

```
} event_info;

RES res;
```

Input

lp

Pointer to the `BTLINE` structure that is attached to the board.

args

Pointer to an argument structure, *args_rtp_event_control*, containing input and output fields.

args.timeout

The time to block, in milliseconds, when waiting for an event. Set to 0 to block indefinitely.

Output

Return values:

0 = The function executed successfully
<0 = An error occurred

args

The *args_rtp_event_control* structure.

args.event_module_num

Packet source module number.

args.event_channel_num

Channel on which the specified event occurred.

args.event_info.event_type

The events reported back to the host are as follows:

Parameter	Additional Data
RTP_EVENT_JITTER_FRAME_OVERDUE	None
RTP_EVENT_JITTER_FRAME_INCORRECT_ORDER	None
RTP_EVENT_JITTER_OVERFLOW	None
RTP_EVENT_RTCP_SRRR	*args.event_info.u.rtcp_data*
RTP_EVENT_RTCP_SDES	*args.event_info.u.rtcp_data*
RTP_EVENT_RTCP_BYE	*args.event_info.u.rtcp_data*
RTP_EVENT_RTCP_APP	*args.event_info.u.rtcp_data*
RTP_EVENT_RTP_PAYLOAD_CHANGE	None
RTP_EVENT_PACKET_STATS	*args.event_info.u.packet_stats*

args.event_info.u.rtcp_data

The *rtp_rtcp* structure

The parameters reported back to the host in *args.event_info.u.rtcp_data*, are as follows

Parameter	Description
version	Version of RTP/RTCP.
pad_flag	Indicates whether RTCP packet contained padding octets.
report_count	Indicates the number of internal parts in report.

packet_length	Indicates the length of RTCP packet in 32-bit words.
packet_type	*rtp_rtcp_type_t* enumerator type, indicating the RTCP packet type: RTCP_SR RTCP_RR RTCP_SDES RTCP_BYE RTCP_APP

args.event_info.u.rtcp_data.u_reports.u_sr

The *u_sr* structure.

Parameter	Description
ssrc	Synchronization source - unique identifier for the RTP session described by this report.
ntp_seconds	Timestamp seconds of the wallclock time, as represented in NTP (Network Time Protocol) format.
ntp_fraction	Fractional element of ntp_seconds.
rtp_timestamp	RTP timestamp.
num_packets_sent	Total number of RTP data packets transmitted by the sender since starting transmission.
num_bytes_sent	Total number of octets transmitted by the sender since starting transmission.
sr_item	Array of receiver report items of type *rtp_rtcp_rr_t* of which there are *report_count*.

args.event_info.u.rtcp_data.u_reports.u_sr.sr_item

The *rtp_rtcp_rr* structure.

Parameter	Description
ssrc	Data source being reported.
frac_lost	Lost packets since last SR/RR/
last_seq	Extended last sequence number received.
jitter	Interarrival jitter

lsr	Last SR packet from this source
dlsr	Delay since last SR packet

args.event_info.u.rtcp_data.u_reports.u_rr

The *u_rr* structure.

Parameter	Description
rr_item	Array of receiver report items of type *rtp_rtcp_rr_t* of which there are *report_count*.
ssrc	Sender generating this report.

args.even_iInfo.u.rtcp_ata.u_reports.u_sdes

The *u_sdes* structure.

Parameter	Description
sdes_chunk	Array of SDES items of type *rtp_rtcp_sdes_chunk*, which contains *report_count* elements.

args.even_info.u.rtcp_data.u_reports.u_sdes.sdes_chunk

The *rtp_rtcp_sdes_chunk* structure.

Parameter	Description
csrc	SSRC/CSRC for this SDES chunk.
total_items	Number of SDES items.
sdes_item	Array of SDES items of type *rtp_rtcp_sdes_item* of which there are *total_items*.

args.even_info.u.rtcp_data.u_reports.u_sdes.sdes_chunk[].sdes_item

The *rtp_rtcp_sdes_item* structure.

Parameter	Description
item_type	SDES item type.
item_length	SDES item length in bytes.
item_text	Non nul-terminated ASCII string describing the SDES item.

`args.event_info.u.rtcp_data.u_reports.u_bye`

The *u_bye* structure

Parameter	Description
ssrc	Array of SSRC identifiers of which there are *report_count*.
reason_length	Length of reason string in bytes
reason	ASCII encoded text string indicating the reason for leaving a session. This string is not nul-terminated and is *reason_length* bytes long.

`args.event_info.u.rtcp_data.u_reports.u_app`

The *u_app* structure.

Parameter	Description
app_data	Application specific data.
app_data_len	Length of application specific data in bytes.

`args.event_info.u.packet_stats`

The *rtp_packet_stats* structure.

Parameter	Description
num_packets	Total number of packets from start.
num_octets	Total number of octets from start.
num_packets_lost	Total number of packets lost from start.
interarrival_jitter	Estimate of the statistical variance of the RTP interarrival time in milliseconds. Detailed algorithm found in RFC1889.
avg_transmission_delay	Average transmission delay. This value is only calculated for full-duplex connections. Estimate of the network latency in milliseconds.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

When *BfvRtpEventGet* returns, the output fields indicate the event type and data content. Different structures are returned in union *u*, depending upon the *event_type*.

The whole RTCP report is limited to 1500 octets. If an RTCP report is received in excess of that (and hence fragmented), the whole report is ignored.

Note: The BTLINE * registered with the BfvRTPEventControl cannot be used for any other bfv function or else the data is discarded.

BfvRtcpReportSend

Purpose

Configures RTCP SDES items and generates application-specific APP reports.

Syntax

```
int
BfvRtcpReportSend          (lp, args)
   BTLINE                  *lp;
   struct args_rtp_event_control *args;
```

The structure contains the following fields.

Input Fields

```
struct rtp_event
{
   rtp_event_type_t event_type;
   union
   {
        struct rtp_rtcp
        {
          rtp_rtcp_type_t packet_type;
          union
          {
               struct sdes_send
               {
                   rtp_rtcp_sdes_type_t sdes_type;
                   char                 sdes_text[255];

               } u_sdes_send;
               struct app
               {
                   char app_data[RTP_MAX_RTCP_APP_DATA];
                   int app_data_len;
               } u_app;
          } u_reports;
        } rtcp_data;
   }u;
} event_info;
```

Output Fields

`RES res;`

Input

lp

Pointer to the `BTLINE` structure that is attached to the board.

args

Pointer to an argument structure, *args_rtp_event_control*, containing input and output fields.

args.event_info.event_type

Selects the type of packet being created, one of:

`RTP_EVENT_RTCP_SDES`

`RTP_EVENT_RTCP_APP`

args.event_info.u.rtcp_data.packet_type

Identifies the RTCP packet, relates to *event_type*, one of:

`RTCP_SDES`

`RTCP_APP`

args.event_info.u.rtcp_data.u_reports.u_sdes_send.sdes_text

This null-terminated ASCII encoded SDES text string is no longer than 255 characters, including the nul character.

args.event_info.u.rtcp_data.u_reports.u_sdes_send.sdes_type

Indicates the type of SDES item:

`RTCP_SDES_END`

`RTCP_SDES_CNAME`

`RTCP_SDES_NAME`

`RTCP_SDES_EMAIL`

`RTCP_SDES_PHONE`

`RTCP_SDES_LOC`

`RTCP_SDES_TOOL`

`RTCP_SDES_NOTE`

`RTCP_SDES_PRIV`

`args.event_info.u.rtcp_data.u_reports.u_app.app_data`

This parameter is dependent upon the RTCP packet type being created. An array of application specific data of maximum size `RTP_MAX_RTCP_APP_DATA` in bytes.

`args.event_info.u.rtcp_data.u_reports.u_app.app_data_len`

This parameter is dependent upon the RTCP packet type being created. Length of application specific data in bytes.

Output

Return values:

0 = The function executed successfully
<0 = An error occurred

`args.res`

A `RES` structure containing status information.

Details

SDES strings must be no greater than 255 characters, including the nul character. The CNAME SDES item is defined to be a unique session identifier. Changing this during a call is not recommended.

If *BfvRtcpReportSend* is called specifying the same *sdes_type* as an existing SDES item, the new SDES item is used. Application specific data must be no greater than 1300 bytes. RTCP reports are intended to provide limited control information. Large or frequent application reports are discouraged to remain within RTCP bandwidth recommendations.

6 - Miscellaneous Functions

This chapter describes a set of useful administration functions which cannot be classified with other functions.

Some administration functions and macros cannot be classified with other functions, but are useful in various ways. For example:

- *_dll_...* functions for use on Windows operating systems. These functions call standard C library functions such as *fopen*, *fclose*, *fread*, and *fwrite* and their arguments use the runtime library linked with the `DLL`.
- The *getopt* function parses command line options in a `Linux` environment. Most of the sample applications/utilities use this function (see *Sample Applications and Utilities* in the Dialogic® Brooktrout® Fax Products SDK *Developer Guide*).
- The *BfvMemAllocFuncsSet* function allows you to write your own functions to dynamically allocate and free memory instead of using the Bfv API functions to do so.
- The `sleep` macro lets you write applications that sleep for a defined period of time (in seconds). This macro is only defined for environments that do not have built-in sleep functions.

Function Summary

Table 9 provides a brief summary of the administration functions used to perform specialized tasks.

Table 9. Miscellaneous Functions Summary

Function	Purpose	Page
dll...	Calls the standard C library function that matches the *_dll_...* function with the arguments provided by using the runtime library linked with the dll.	*211*
BfvGetVar	Requests the value of a specified facility firmware variable.	*212*
BfvLineAlert	Interrupts an active channel for another use by suspending, but not killing, the interrupted process or thread.	*215*
BfvMemAllocFuncsSet	Replaces Bfv API functions that dynamically allocate and free memory with applications' functions that do the same.	*218*
BfvRcvProcessPkt	Receives a packet, and performs internal Bfv API processing of all commands contained within the packet.	*221*
BfvSetSingleVar	Attempts to send a SET command to set a single variable.	*224*
getopt	Parses command line options.	*228*

dll...

Purpose

Calls the standard C library function that matches the *_dll_*... function with the arguments provided by using the runtime library linked with the dll.

Syntax

One of the following:

```
_dll_fopen
_dll_fclose
_dll_fread
_dll_fwrite
_dll_free
_dll_fseek
_dll_ftell
_dll_malloc
_dll_stdout
_dll_stdin
_dll_stderr
```

Input

Same as the standard C library functions. See your compiler manual for details.

Output

Same as the standard C library functions. See your compiler manual for details.

Details

Each of these functions takes the same arguments and has the same return type as the standard C library function that has the same name without the *_dll_*... prefix. See your compiler manual for information about these functions. Each *_dll_*... function calls the matching standard C library function with the arguments provided by using the runtime library linked with the `DLL`.

Applications use these functions on platforms where Dialogic supplies `DLL` versions of the Bfv API library. These platforms include Windows.

BfvGetVar

Purpose

Requests the value of a specified facility firmware variable.

Syntax

```
void
BfvGetVar                    (lp, args)
BTLINE                       *lp;
struct args_packet           *args;
```

The structure contains the following fields.

Input Fields

```
unsigned facility;
unsigned tag_id;
unsigned char *pkt_buf;
unsigned pkt_len;
int no_hangup;
int use_alt_chan;
BTLINE *async_lp;
```

Output Fields

```
unsigned tag_len;
unsigned tag_type;
unsigned tag_data_len;
unsigned tag_data_ptr;
unsigned var_value;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.facility

Facility of the variable to retrieve. See the command set for possible values.

args.tag_id

Tag identifier of the variable. See the command set for possible values.

args.pkt_buf

Optional packet buffer.

args.pkt_len

Size of the optional packet buffer.

args.no_hangup

Unless nonzero, line hangup or equivalent conditions are treated as an error.

args.use_alt_chan

If nonzero, specifies an alternate channel within the default module to use for sending commands.

args.async_lp

If non-`NULL`, enables async operation. The variable contains the line pointer of a session to be notified when this operation is completed. See the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information on async operation and usage.

Output

Return value: None.

args.tag_len

Length of the tag returned.

args.tag_type

Type of the tag returned.

args.tag_data_len

Data length of the tag returned.

args.tag_data_ptr

Data pointer for the tag returned.

args.var_value

Value of the variable, if integer and not array.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function should only be used under the guidance of Dialogic Technical Services and Support.

The function requests the value of a facility firmware variable specified by *args.facility* and *args.tag_id*. If the variable is of an integer type and is only a single element (not an array), *args.var_value* will be set to the value of the variable.

The packet is stored in a fixed internal buffer unless *args.pkt_buf* is supplied.

If the value of *args.timeout* is `MILL_MAX_TIMEOUT`, which is the maximum value of an unsigned integer, then the user configuration file (*btcall.cfg*) parameter *max_timeout* applies to the timeout.

BfvLineAlert

Purpose

Interrupts an active channel for another use by suspending, but not killing, the interrupted process or thread.

Syntax

```
void
BfvLineAlert                (args)
   struct args_line_admin  *args;
```

The structure contains the following fields.

Input Fields

```
int unit;
unsigned char alert_value;
MILL_ADDR dest_addr;
MILL_ADDR local_addr;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.unit

Number of the channel whose associated application receives the alert.

–1 = all channels.

args.alert_value

An alert value to be passed to the alerted channel. If left set to 0, a value of 1 will be sent.

args.dest_addr

Specifies the address of the active channel that the Bfv API notifies of its intention to temporarily interrupt for another use. This is the address of a channel on a board. The channel and module values must be set as appropriate. The facility value must be set to `M_ADDR_WILDCARD`. The machine value can be set to 0 for the current machine, or set as appropriate.

`args.local_addr`

Explicitly specifies the address of the application receiving the alert for a temporary interruption. The facility, channel, and module values must be set as appropriate. The machine value can be set to 0 for the current machine, or set as appropriate.

Output

Return value: None

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Interrupts an application actively using a channel for another use without killing the interrupted process or thread and without introducing dangers caused by using *longjmp*.

This function sends an “alert” to the application associated with the channel specified by *args.unit* or all channels if *args.unit* is –1. The application session to alert can also be specified explicitly using *args.local_addr* to give its address or implicitly using *args.dest_addr* to give the associated address.

Alerting an application associated with a channel causes a subsequent call to low level packet functions including packet processing to:

- Indicate that a packet was received
- Temporarily change the line state to `IDLE`
- Set other internal line structure fields to distinguish the alert from other errors.

When an application associated with a channel receives an “alert”, any Bfv API function called on the channel will return. If the function uses a `RES` structure, a `res.status` of `BT_STATUS_ALERT` is returned. The function will also return an error indication if it has that capability.

This function is not intended for interprocess or interthread communications, but rather for aborting operations already in progress on a channel.

You can use this function, for example, to temporarily interrupt a channel that is waiting indefinitely for a call to arrive in order to transmit a fax, or to interrupt a fax transmission or any other operation taking a significant amount of time.

Normally, an alerted channel calls the *BfvLineReset* function.

Using the macro `LINE_ALERT_CTL` and the macro `LINE_SET_INCOMING_CMD_FUNC`, the application can control its own behavior when a channel receives an alert, and can make use of the alert value passed. See *Macros on page 82* for more information about these macros.

If using multiple application sessions attached to the same channel or destination address, alerting based on that channel or address will not reliably alert the desired application session. Alerting based on the target application session's address is recommended.

This function uses one application session, which is normally only for a very short span of time on an infrequent basis, and the session is freed before the function returns.

BfvLineAlert must be called from a separate thread or process from that used by the channel that is to be interrupted

See Also

`LINE_ALERT_CTL`, `LINE_SET_INCOMING_CMD_FUNC`

Example

```
/* Alert channel 0 */
struct args_line_admin args;

BT_ZERO(args);
args.unit = 0;
BfvLineAlert(&args);
```

BfvMemAllocFuncsSet

Purpose

Replaces the Bfv API functions that dynamically allocate and free memory with functions the application provides.

Syntax

```
void
BfvMemAllocFuncsSet   (alloc_func_ptr, free_func_ptr)
   void               * (*alloc_func_ptr)
                      (unsigned size,
                      int channel, int mem_type);
   void               (*free_func_ptr)
                      (void *ptr, int channel,
                      int mem_type);
```

Input

alloc_func_ptr

A pointer to a user-supplied function that the Bfv API calls when it needs to dynamically allocate memory.

Alloc_func_ptr is called as:
`(*alloc_func_ptr)(size,channel,mem_type)`

The *size* argument contains the number of bytes of memory to allocate; see this function's *Details* section for a description of the *channel* and *mem_type* arguments.

The user-supplied function must return a valid pointer to the allocated memory or `NULL` if the allocation fails.

free_func_ptr

A pointer to a user-supplied function that the Bfv API calls when it needs to free memory it dynamically allocated previously.

Free_func_ptr is called as:
`(*free_func_ptr)(ptr,channel,mem_type)`

The *ptr* argument contains the pointer to the memory to free; see this function's *Details* section for a description of the *channel* and *mem_type* arguments.

Output

Return value: None.

Details

This function gives applications complete control over dynamic memory allocation on a per-type basis, permitting them to use a number of strategies, such as allocating memory from a fixed-size static pool.

When used in a multithreaded environment, this function applies to all threads and to all channels.

The Bfv API performs all dynamic memory allocation, including structures it not only explicitly allocates, but also the buffers that the *stdio* ***fread*** and ***fwrite*** functions use, using the user-supplied functions.

The Bfv API calls the allocation and free functions with *channel* and *mem_type* arguments. The value of *channel* is either the number of the associated channel (when appropriate and available) or –1. For example, an associated channel number is available when attaching a channel and when allocating `PAGE_RES` structures, but no channel number is available when opening an infopkt file or when opening a TIFF file. The possible values of *mem_type* are given by the `BTMEM_...` symbols defined in the *mill_api.h* header file.

If the application does not call *BfvMemAllocFuncsSet*, the Bfv API uses a default set of memory allocation and free functions that simply call ***malloc*** and ***free***, respectively.

The user-supplied functions are called only when Bfv API functions allocate or free memory. Neither the application nor the non-Bfv API functions called by the application use the user-supplied functions to allocate memory.

The user-supplied function must not call any function that causes a delay, such as waiting for a `DTMF` tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing `DTMF` tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

In multithreaded environments, make sure that the allocation and free functions are re-entrant. Standard C library *malloc* and *free* functions are already re-entrant. If the application uses a static allocation strategy, it might need a synchronization object.

Example

```
void *allocfunc(n, c, t)
unsigned n;
int c, t;
{
   void *ptr = (void *)malloc(n);

   printf("allocated %u bytes, chan %d, type %d\n",
   n, c, t);
   return (ptr);
}

void freefunc(ptr, c, t)
void *ptr;
int c, t;
{
   free(ptr);
   print("Freed memory chan %d, type %d\n", c, t);
}

main()
{
   ...
   BfvMemAllocFuncsSet(allocfunc, freefunc);
   ...
}
```

BfvRcvProcessPkt

Purpose

Attempts to receive a packet, and performs internal Bfv API processing of all commands contained within the packet.

Syntax

```
int
BfvRcvProcessPkt            (lp, args)
   BTLINE                   *lp;
   struct args_packet       *args;
```

The structure contains the following fields.

Input Fields

```
unsigned timeout;
int no_hangup;
unsigned char *pkt_buf;
unsigned pkt_len;
int ignore_data;
BTLINE *async_lp;
```

Output Fields

```
RES res;
```

Modified Fields

pkt_buf, pkt_len, cmd_buf, cmd_len, sent, flags, dest_addr, src_addr, facility.

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.timeout

Time to wait for an incoming packet in milliseconds.

args.no_hangup

Unless nonzero, line hangup or an equivalent condition is treated as an error.

`args.pkt_buf`

Optional packet buffer. It is recommended that the buffer be large enough to store the maximum packet size of 1K.

`args.pkt_len`

Size of the optional packet buffer.

`args.ignore_data`

If nonzero, DATA commands are ignored during packet processing.

`args.async_lp`

If non-`NULL`, async operation is enabled. The variable contains the line pointer of a session to be notified when this operation is completed. See the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information on async operation and usage.

Output

Return value: Indicates status

`MILL_RCV_COMPL`

A packet was successfully received.

`MILL_RCV_ERR`

An error occurred while attempting to receive.

Values can be logically ORed together.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function should only be used under the guidance of Dialogic Technical Services and Support.

The packet is stored in a fixed internal buffer that is at least 1K in size, unless *args.pkt_buf* is supplied. Waits for up to *args.timeout* milliseconds to receive a packet.

If the value of *args.timeout* is `MILL_MAX_TIMEOUT`, which is the maximum value of an unsigned integer, then the user configuration file (*btcall.cfg*) parameter *max_timeout* applies to the timeout.

Example

```
BTLINE *lp;
struct args_packet args;
int tmp;

/* Wait for a packet for up to 10 seconds. */
BT_ZERO(args);
args.timeout = 10000;
tmp = BfvRcvProcessPkt(lp,&args);

if (tmp & MILL_RCV_COMPL)
   printf("Packet received.\n");
if (tmp & MILL_RCV_ERR)
   printf("Error Receiving packet.\n");
```

BfvSetSingleVar

Purpose

Attempts to send a SET command to set a single variable.

Syntax

```
int
BfvSetSingleVar              (lp, args)
   BTLINE                    *lp;
   struct args_packet        *args;
```

The structure contains the following fields.

Input Fields

```
unsigned facility;
unsigned tag_id;
unsigned tag_type;
unsigned tag_data_len;
unsigned char *tag_data_ptr;
unsigned var_value;
int prio;
int incoming_flag;
unsigned timeout;
int no_hangup;
int use_alt_chan;
BTLINE *async_lp;
```

Output Fields

```
RES res;
```

Modified Fields

cmd_buf, cmd_verb, cmd_specifier, tag_id, tag_type, tag_data_len, tag_data_ptr, dest_addr, src_addr, flags, cmd_len, pkt_buf, pkt_len, tag_ptr, tag_len.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

`args.facility`

Facility to send to on the associated channel. See the command set for possible values.

`args.tag_id`

Tag ID of the variable to set. See the command set for possible values.

`args.tag_type`

Tag type of the variable to set. See the command set for possible values.

`args.tag_data_len`

Length of the tag data specified by *args.tag_data_ptr*. If 0, the tag data must be a single integer value and is specified by *args.var_value*.

`args.tag_data_ptr`

Pointer to the tag data if *args.tag_data_len* is nonzero.

`args.var_value`

Tag data value if *args.tag_data_len* is 0.

`args.prio`

Specifies the priority of the packet:

`MILL_PKT_PRIO_LOW`	0	Low (normal) priority
`MILL_PKT_PRIO_HIGH`	1	High priority

`args.incoming_flag`

Affects the behavior when incoming packets arrive:

`MILL_SEND_INCOMING_IGNORE`	Ignore incoming packets.
`MILL_SEND_INCOMING_RETURN`	Return if an incoming packet is available.

MILL_SEND_INCOMING_PROCESS	Process incoming packets while attempting to send. Returns when send completes or an error occurs.
MILL_SEND_INCOMING_PROCESS_RETURN	Process incoming packets while attempting to send. Return if an incoming packet arrives.

args.timeout

Time, in milliseconds, to wait to queue the packet.

args.no_hangup

Unless nonzero, line hangup or an equivalent condition is treated as an error.

args.use_alt_chan

If nonzero, specifies an alternate channel within the default module to use for sending commands.

args.async_lp

If non-NULL, enables async operation. The variable contains the line pointer of a session to be notified when this operation is completed. See the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information on async operation and usage.

Output

Return value: Indicates status as:

MILL_SEND_COMPL	The packet was successfully queued.
MILL_SEND_ERR	An error occurred while attempting to send.
MILL_SEND_INCOMING	An incoming packet arrived.
MILL_SEND_INC_PROC	An incoming packet was processed.

Values can be logically ORed together.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function should only be used under the guidance of Dialogic Technical Services and Support.

The Bfv API forms the packet addressing using information from *BfvSessionAttach* or *BfvLineAttach*, *args.facility*, and priority specified by *args.prio*. Waits for up to *args.timeout* milliseconds to queue the packet.

If the value of *args.timeout* is `MILL_MAX_TIMEOUT`, which is the maximum value of an unsigned integer, then the user configuration file (*btcall.cfg*) parameter *max_timeout* applies to the timeout.

See Also

BfvSendCmd

getopt

Purpose

Parses command line options.

Syntax

```
int
getopt                          argc, argv, optstring)
   int                          argc;
   char                         **argv;
   char                         *optstring;
```

Input

argc

The *argc* value from main.

argv

The *argv* value from main.

optstring

A string that specifies what options the program accepts and whether or not the options take arguments. All of the option letters must appear, and if they take an option argument, they must be followed by a colon (:).

Output

Return value:

>0 Identifies the next option letter in *argv* that matches a letter in *opstring*. The function modifies *optarg* to point to the option argument.

'?' Indicates that an unexpected option letter appeared or an option was missing an option argument. The function also prints an error message on *stderr* unless the application sets *opterr* to 0.

–1 Indicates that the function processed all options and that only operands remain.

In all cases, the function modifies *optind* to contain the index in *argv* of the next command line argument to be processed.

Details

Command line entries must adhere to this format:

```
<cmd> –<optlet1> –<optlet2> <arg2> <operands>
```

All options are single letters preceded by a hyphen (-). An option may take an option argument. The option argument always follows the option letter, and white space separates the option letter and its option argument. Particular options either always or never take an option argument.

Three global variables are associated with getopt:

```
extern int optind;
extern char *optarg;
extern int opterr;
```

This function is standard under Linux. In all other environments, the Bfv API provides this function to applications. Most of the sample and utility programs found in the *app.src* directory use this function.

For a complete description of this function, see the Linux programmer's reference manuals. The functionality described here, however, is all that the programs in the *app.src* directory require.

Example

See applications in the sample application directory.

7 - Debugging, Error Handling and Return Values

This chapter describes functions to assist in debugging problems and recovering from errors, as well as describing the levels and types of logging available.

Dialogic provides several Bfv API functions to help you debug your application program and find and recover from errors.

You can turn on debug mode so that the Bfv API prints commands, data, and status messages, or you can set up a function to be used with API debug mode that directs output to a file or filter. See *Debugging* in *Chapter 3* of the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information.

When you install the Bfv API, you enable recording of the history of the activity of the driver along with the hardware type, the firmware version, and the boot ROM version. You can then use functions to dump the buffer containing the driver's history for a module and channel to a file. You can also clear the history buffer for a module and channel so that it will contain information relevant to the current application.

If you have a `RES` structure that contains returned error information from a previous Bfv API call, you can use the *BfvErrorMessage* function to create a short and a long error message in a `BTERR` structure. The application can then choose to print the short or long message returned by the function in this structure.

Dialogic provides a Call Tracer command line utility that collects call trace information in an active system. The output is intended for Dialogic Technical Services and Support, but it is important that all users know how to use Call Tracer to create the output file, if Dialogic Technical Services and Support personnel request it. The Call Tracer utility can be started before or after starting the client application. If you want to trace the initialization section of the client application, start the Call Tracer before the client application.

For information on how to start the Call Tracer, type brktcctrace -?. Exit the Call Tracer application by typing ‘q’ or ‘Control-C’, or by closing the command console window. The Call Tracer application reads trace filter settings from a text configuration file called *filtersettings.cfg*. The output is logged to a file name of your choosing.

For log information internal to the Call Tracer, the application maintains its own log file that is located in the current working directory of the application. The tracer logs all warning, error and panic level messages by default.

The Call Tracer utility, and a sample configuration file, can be found in the *\Brooktrout\Boston\utils\winnt\bin* directory when installing the Brooktrout SDK, or in the *\Brooktrout\bin* directory when installing just the System Software. See *Debugging* in *Chapter 3* of the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for instructions on how to run the Call Tracer application.

Structures and Return Values

The Bfv API uses argument structures to pass values to and from functions. The argument structure is declared in an application and passed as a pointer to the function. The argument structure type will be named *args_...*; for example, *struct args_fax*. The same argument structure type is used for functions that are related or in the same category.

Contained within the argument structure are structure fields that are used for input and/or output. Each function that uses an argument structure has the fields marked that are used for each purpose. Not all fields are used by all functions taking any particular argument structure type.

Result structures are the most commonly used structures to return information to the function. They are:

- *RES* structure – returns status information in *res.status* and some additional information in *res.line_status*.
- *CALL_RES* – returns information about a call, such as its type and caller ID. If applicable, ISDN information, such as called party and redirect information, are returned as well.
- *PAGE_RES* – returns information for each complete fax page sent or received.

For more information about the result structures, see *Volume 6, Appendix B, Bfv API Structures.*

Function Summary

Table 10 provides a brief summary of the functions to use for debugging and error handling purposes.

Table 10. Debugging and Error Handling Function Summary

Function	Purpose	Page
BfvDebugFuncSet	Sets up a function to use with Bfv API debug mode that directs debug output to an alternate destination.	*234*
BfvDebugInitData	Recreates name tables used for Bfv API debug mode and the *dh* program, based on command set header files found in a specified directory.	*236*
BfvDebugModeSet	Enables debug mode, so the Bfv API prints commands, data, interrupts, and status messages to the standard output or alternate device.	*237*
BfvDebugModeSetAdv	Allows the application to configure Bfv API debugging features within the application, enabling the user to control debugging in a remote application.	*239*
BfvErrorMessage	Returns error message strings corresponding to Bfv API errors returned in `RES` structures.	*244*
BfvHistoryClear	Clears the driver's history buffer.	*246*
BfvHistoryClearModChan	Clears the contents of the driver's history buffers for the specified module and channel number.	*248*
BfvHistoryClearUnit	Clears the contents of the driver's history buffers on the channel specified by the channel number.	*250*
BfvHistoryDump	Dumps the driver's history buffer to the specified open file.	*252*
BfvHistoryDumpModChan	Dumps the contents of the driver's history buffer for the specified module and channel number to the specified open file.	*255*
BfvHistoryDumpUnit	Dumps the contents of the driver's history buffer specified by the channel number to the specified open file.	*258*
BfvLineDumpStructure	Dumps the contents of the `BTLINE` structure to the specified open file.	*261*

BfvDebugFuncSet

Purpose

Sets up a function to use with Bfv API debug mode that directs debug output to an alternate destination — a file, filter, or non-*stdio* device.

Syntax

```
void
BfvDebugFuncSet                (func)
   void                        (*func)  (char  *msg);
```

Input Fields

```
unsigned enable;
unsigned notify_type;
void (*func)(BTLINE *lp, unsigned notify_type,
   struct args_board_notify *args);
```

Output Fields

```
RES res;
```

Input

args.func

A pointer to a user-supplied function that the Bfv API will call when it has an Bfv API debug mode message to display. `Func` will be called as `(*func)(msg)` where `msg` contains the message to print.

Output

Return value: None.

Details

When used in a multithreaded environment, this function applies to all threads and to all channels.

The *BfvDebugModeSet* function enables Bfv API debug mode output.

The *msg* argument passed to the user-supplied function will be 0-terminated, but generally it will not contain a “newline” character. The function is responsible for adding “newline” characters when appropriate.

The user-supplied function can direct the output a variety of ways, including storing it in a file, displaying it in a non-*stdio* way, or filtering it.

If the application does not call *BfvDebugFuncSet*, the Bfv API uses a default output function. This function calls *puts* first to display the output on *stdout* and then *fflush*.

The Bfv API can detect a number of fatal errors that will produce debug mode output, even if no debug output has been enabled. Therefore, use *BfvDebugFuncSet* to set up an appropriate function if ordinary *printf* and *fflush* function calls to *stdout* will cause problems in your operating environment.

The user-supplied function must not call any function that causes a delay, such as waiting for a DTMF tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing DTMF tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

See *Debugging* in *Chapter 3* of the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information.

See Also

BfvDebugModeSet

Example

```
FILE *fp;

void dfunc(msg)
char *msg;
{
    fprintf(fp, "%s\n", msg);
}

main()
{
    ...
    BfvDebugModeSet(DEBUG_ALL);
    fp = fopen("debug.log", "w");
    BfvDebugFuncSet(dfunc);
    ...
}
```

BfvDebugInitData

Purpose

Recreates name tables used for Bfv API debug mode and the *dh* program, based on command set header files found in a specified directory.

Syntax

```
void
BfvDebugInitData            (dir_name)
   char                     *dir_name;
```

Input

dir_name

Directory storing the command set header files to read.

Output

Return value: None.

Details

Use this function if modifications were made to commands, or additional command header files are made available after compilation and distribution of the program. This function provides the location of new unreleased command set header files so that debug mode functions can understand the command additions. You do not need to recompile the Bfv API.

Infrequently used in normal operation.

This function can only be called once per process.

Example

```
BfvDebugInitData("/tmp/testfiles");
```

BfvDebugModeSet

Purpose

Enables debug mode, so the Bfv API prints commands, data, interrupts, and status messages to the standard output or alternate device.

Syntax

```
int
BfvDebugModeSet          (mode)
   int                   mode;
```

Input

mode

A value that indicates whether debug mode is enabled or disabled and sets the form the debug output will take.

The most frequently used values are:

DEBUG_NONE 0

No output.

DEBUG_ALL -1

Print all debug messages.

Logically OR together the following values to form other *mode* values:

DEBUG_PRINT_CMD 0x01

Print information about commands sent from the Bfv API to the firmware by the driver.

DEBUG_PRINT_INTR 0x02

Print information about commands sent from the firmware to the Bfv API by the driver.

DEBUG_MON 0x04

Print informational and status messages.

`DEBUG_ERR` 0x08

Print error messages.

`DEBUG_DEBUG` 0x10

Print output from the firmware *debug* interrupt (not currently used).

Output

Return value: None.

Details

Debug mode can produce a large amount of output that can affect system performance under heavy usage conditions.

By default, the Bfv API prints debug output on the standard output. To change this behavior, use the *BfvDebugFuncSet* function.

Bfv API debug mode is disabled by default (0).

The output this function enables is separate and independent of the dump history output that the *dh* program and the *BfvHistoryDump*... functions produce.

See *Debugging* in *Chapter 3* of the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information.

When used in a multithreaded environment, this function applies to all threads and to all channels.

See Also

BfvDebugFuncSet

Example

```
main()
{
    ...
    if (verbose_option)
        BfvDebugModeSet(DEBUG_ALL);
    else
        BfvDebugModeSet(DEBUG_NONE);
    ...
}
```

BfvDebugModeSetAdv

Purpose

Allows the application to configure Bfv API debugging features within the application, enabling the user to control debugging in a remote application.

Syntax

```
void
BfvDebugModeSetAdv          (args)
   struct args_debug_mode  *args;
```

The structure contains the following fields.

Input Fields

```
unsigned options;
M_CB1 void (* M_CB2 func) (char *msg);
int mode;
char *fname1;
char *fname2;
unsigned file_limit;
int unit;
MILL_ADDR dest_addr;
MILL_ADDR local_addr;
int cc_trace_level;
char *cc_trace_file_name;
int func_dbg_opts;
```

Output Fields

```
RES res;
```

Input

args.options

A bit-mapped value indicating the debug features to select. Multiple options can be ORed together.

Defined options are as follows:

API_DBG_EXT_CTRL	0x00000001
API_DBG_MODE_SET	0x00000002
API_DBG_FUNC_SET	0x00000004
API_DBG_FUNC_ENTRY_EXIT	0x00000008
API_DBG_CC_API_SET	0x00000010

API_DBG_CC_L3L4_SET	0x00000020
API_DBG_CC_L4L3_SET	0x00000040
API_DBG_CC_INT_SET	0x00000080
API_DBG_CC_HOST_MOD_SET	0x00000100
API_DBG_CC_IP_STACK_SET	0x00000200
API_DBG_CC_LOG_FILE_SET	0x00000400
API_DBG_FILE_SET	0x00000800

args.func

A pointer to a user-supplied function that the API will call when it has an Bfv API debug mode message to display. The `func` variable will be called as `(*func)(msg)` where `msg` contains the message to print.

args.mode

A value that indicates whether debug mode is enabled or disabled and sets the form the debug output will take.

When the `API_DBG_MODE_SET` bit is set in the *options* field this *mode* field has the same meaning as the *mode* field in the *BfvDebugModeSet* function as detailed below.

The most frequently used values are:

`DEBUG_NONE` 0

No output.

`DEBUG_ALL` -1

Print all debug messages.

Logically OR together the following values to form other *mode* values:

`DEBUG_PRINT_CMD` 0x01

Print information about commands sent from the Bfv API to the firmware by the driver.

`DEBUG_PRINT_INTR` 0x02

Print information about commands sent from the firmware to the Bfv API by the driver.

`DEBUG_MON` 0x04

Print informational and status messages.

`DEBUG_ERR` 0x08

Print error messages.

`DEBUG_DEBUG` 0x10

Print output from the firmware *debug* interrupt (not currently used).

`args.fname1`

Set to `NULL` to direct debug output to `STDOUT`. Otherwise, set to the full path and filename of the file that the debug output will be written to. Only used if *args.options* has `API_DBG_FILE_SET` bit set, and no debug function has *args.options* `API_DBG_FUNC_SET` bit set.

`args.fname2`

If fname1 is non-`NULL`, this parameter is the full path and filename of the second file that the debug output will be written to. Note: *args.file_limit* must be defined for this feature to work. Only used if *args.options* has `API_DBG_FILE_SET` bit set, and no debug function has *args.options* `API_DBG_FUNC_SET` bit set.

`args.file_limit`

The maximum length which the debug file should be allowed to grow to. Only used if *args.options* has `API_DBG_FILE_SET` bit set, and no debug function has *args.options* `API_DBG_FUNC_SET` bit set.

`args.unit`

`args.dest_addr`

`args.local_addr`

When the `API_DBG_EXT_CTRL` bit is set in the options field, these fields allow the user to specify a channel in a remote application to change the debug settings on. If the unit is set to -1, the debug command will apply to the current application.

Note that all channels in the application specified by this parameter will have debugging enabled.

The meanings of *unit*, *args.dest_addr*, and *args.local_addr* have the same meanings as for the BfvLineAlert function.

`args.cc_trace_level`

The level at which to set call control tracing.

Valid levels are `ecc_trace_none, ecc_trace_error, ecc_trace_warning, ecc_trace_basic, ecc_trace_verbose.`

`args.cc_trace_file_name`

The full path and filename to output call control tracing to.

`args.func_dbg_opts`

When *args.options* includes `API_DBG_FUNC_ENTRY_EXIT`, this field specifies whether the function entry/exit debugging feature is enabled, and in what mode. The value is a bitmapped value.

Defined options:

```
API_DBG_FUNC_OPT_ENA    0x00000001
API_DBG_FUNC_OPT_ARGS   0x00000002
```

If `API_DBG_FUNC_OPT_ENA` appears, the entry/exit debugging feature will be enabled. Otherwise it is disabled.

If `API_DBG_FUNC_OPT_ARGS` appears, the debugging will also include argument values and return values.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The following details the function behavior for the various *options*:

Setting `API_DBG_EXT_CTRL` allows the debugging in another application to be controlled via *unit*, *args.dest_addr* and *args.local_addr*.

Setting `API_DBG_MODE_SET` allows the mode field to set the debug output of the Bfv API.

Setting `API_DBG_FILE_SET` allows *args.fname1* and *args.fname2* to set debug filenames, and *args.file_limit* to set a debug file size limit.

Setting `API_DBG_FUNC_SET` allows the func field to set the debug callback function.

Setting `API_DBG_FUNC_ENTRY_EXIT` allows control of Bfv tracing the entry and exit of all Bfv API calls. The value of *args.func_dbg_opts* controls whether the feature is enabled, and in what mode.

Setting `API_DBG_CC_API_SET` causes Bfv to trace the entry and exit of all call control Bfv API calls to the trace level specified in *args.cc_trace_level*.

Setting `API_DBG_CC_L3L4_SET` turns on `L3L4` call control tracing to the trace level specified in *args.cc_trace_level*.

Setting `API_DBG_CC_L4L3_SET` turns on `L4L3` call control tracing to the trace level specified in *args.cc_trace_level*.

Setting `API_DBG_CC_INT_SET` turns on internal call control tracing to the trace level specified in *args.cc_trace_level*.

Setting `API_DBG_CC_HOST_MOD_SET` turns on Host Module call control tracing to the trace level specified in *args.cc_trace_level*.

Setting `API_DBG_CC_IP_STACK_SET` turns on IP call control tracing to the trace level specified in *args.cc_trace_level*.

Setting `API_DBG_CC_LOG_FILE_SET` sets the name of the call control log file to *args.cc_trace_file_name*. If NULL, the current log file is closed and no log file is used.

See *Debugging* in *Chapter 3* of the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information.

Certain call control messages might not appear in the call control trace file when enabled using this function. However, these messages will appear in the Bfv API debug log.

See Also

BfvDebugFuncSet, *BfvDebugModeSet*

Example

See the *debug_control* application in the *bapp.src* sample applications directory.

BfvErrorMessage

Purpose

Returns error message strings corresponding to Bfv API errors returned in RES structures.

Syntax

```
void
BfvErrorMessage          lp, res, err_msg)
   BTLINE                *lp;
   RES                   *res;
   BTERR                 *err_msg;
```

Input

lp

Pointer to the **BTLINE** structure.

Can be NULL, in which case the error messages might not have complete information.

err_msg

Pointer to a preallocated structure of type BTERR to store the formatted error messages.

res

Pointer to a RES structure containing returned error information from a previous Bfv API call.

Output

Return value: None.

Returns the following modified structure pointed to by the argument *err_msg*:

```
typedef struct {
    char short_msg[31];
    char long_msg[129];
} BTERR;
```

Details

This function returns a pair of textual error message strings corresponding to an Bfv API error returned in a RES structure.

Returns both a short and long error message. The function copies the messages into a BTERR structure that the application passes in.

Example

```
BTLINE *lp;
struct infopkt_stream *ips;
BTERR bterr;
struct args_fax args;

BT_ZERO(args);
args.s_ips = ips;
args.local_id = "my_id";
BfvFaxSend(lp, &args);

if (args.res.status != BT_STATUS_OK)
{
    BfvErrorMessage(lp, &args.res, &bterr);
    printf("Fax sending failed: %s\n", bterr.long_msg);
}
```

BfvHistoryClear

Purpose

Clears the contents of the driver's history buffers on the specified channel.

Syntax

```
void
BfvHistoryClear              (lp, args)
   BTLINE                    *lp;
   struct args_dh            *args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

RES *res;*

Modified Fields

unit, module, channel.

Input

lp

Pointer to the **BTLINE** structure containing the channel to clear its history.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Clears the contents of the driver's history buffers on the channel specified by the **BTLINE** * line pointer.

Use this function to ensure that a later *BfvHistoryDump*... call contains only information relevant to an application about to run.

The application can specify a line pointer, a module and channel, a channel, or all channels.

Clears the history for the attached destination module and channel, if a history was created for those values. Typically, an application calls the *BfvHistoryClearModChan* with the values of the *args.module* and *args.channel* fields set to 1 to clear the module and channel buffers.

Once the application clears the history buffer, it cannot recover the discarded information.

See Also

BfvHistoryDump...

Example

```
main()
{
    int unit;
    struct args_dh args;
    BTLINE *lp = BfvLineAttach(unit);
    ...
    BT_ZERO(args);
    BfvHistoryClear(lp, &args);
    ...
}
```

BfvHistoryClearModChan

Purpose

Clears the contents of the driver's history buffers for the specified module and channel number.

Syntax

```
void
BfvHistoryClearModChan      (args)
   struct args_dh           *args;
```

The structure contains the following fields.

Input Fields

```
unsigned int module;
unsigned int channel;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.module

The number of the module to clear its history. Usually set to 1.

args.channel

The number of the channel to clear its history. Usually set to 1.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this function to ensure that a later *BfvHistoryDump*... call contains only information relevant to an application about to run.

When configured to include application histories, there are some special module meanings. Set *args.module* to 0 to use the history for the most recent application session corresponding to *args.channel* as unit/ordinal channel number. Set *args.module* to 0xFE to use the fixed application history corresponding to *args.channel* as an index value.

Once the application clears the history buffer, it cannot recover the discarded information.

See Also

BfvHistoryDump...

Example

```
main()
{
    struct args_dh args;
    ...
    BT_ZERO(args);
    args.module = 1;
    args.channel = 1;
    BfvHistoryClearModChan(&args);
    ...
}
```

BfvHistoryClearUnit

Purpose

Clears the contents of the driver's history buffers on the channel specified by the channel number.

Syntax

```
void
BfvHistoryClearUnit          (args)
   struct args_dh            *args;
```

The structure contains the following fields.

Input Fields

int *unit;*

Output Fields

RES *res;*

Modified Fields

module, channel.

Input

args

Pointer to an argument structure containing input and output fields.

args.unit

The number of the channel to clear its history.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this function to ensure that a later *BfvHistoryDump*... call contains only information relevant to an application about to run.

Clears the history for the destination module and channel associated with the ordinal channel value specified by *unit*, if a history was created for those values. Typically, an application calls the *BfvHistoryClearModChan* with the values of the *args.module* and *args.channel* fields set to 1 to clear the module and channel buffers.

Once the application clears the history buffer, it cannot recover the discarded information.

See Also

BfvHistoryDump...

Example

```
main()
{
    int unit;
    struct args_dh args;
    ...
    BT_ZERO(args);
    args.unit = unit;
    BfvHistoryClearUnit(&args);
    ...
}
```

BfvHistoryDump

Purpose

Dumps the driver's history buffer for the specified channel to the specified open file.

Syntax

```
void
BfvHistoryDump              (lp, args)
   BTLINE                   *lp;
   struct args_dh           *args;
```

The structure contains the following fields.

Input Fields

```
FILE *fp;
int continuous_output;
int raw_dump;
```

Output Fields

```
RES res;
```

Modified Fields

unit, module, channel.

Input

lp

Pointer to the **BTLINE** structure containing the channel history to dump.

args

Pointer to an argument structure containing input and output fields.

args.fp

FILE * pointer to an open file to receive the channel's history.

args.continuous_output

When set to 1, output displays continuously (similar to tail -f) until the process is killed. During the life of the process, dump history repeatedly prints all available new history information and sleeps for 1/10 sec. History entries can be lost if activity is rapid enough to exceed the capacity of the driver history buffer during the sleep period or other periods of process inactivity.

`args.raw_dump`

When set to 1, the output is an ASCII representation of the raw history data, uninterpreted.

When set to 2, the output is the raw binary history data.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Dumps the contents of the driver's history buffer for the channel specified by the **`BTLINE`** * line pointer to the specified open file.

Dumps the history for the attached destination module and channel that only produce output if a history was created for those values. Typically, an application calls the *BfvHistoryDumpModChan* with the values of the *args.module* and *args.channel* fields set to 1 to dump the contents of the module and channel buffers.

The history output contains additional information, including the time the history was created, driver version, and operating system platform.

The dump history output that these *BfvHistoryDump*... functions produce is separate and independent of the Bfv API debug mode output that the *BfvDebugModeSet* function enables.

See *Debugging* in *Chapter 3* of the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information.

See Also

BfvHistoryClear...

Example

```
main()
{
    int unit;
    struct args_dh args;
    BTLINE *lp = BfvLineAttach(unit);
    FILE *f = fopen ("dh.log", "w");

    ...
    BT_ZERO(args);
    args.fp = f;
      BfvHistoryDump(lp,&args);
    ...)
```

Note: When using Windows, it may be necessary to substitute "fopen" with "_dll_fopen" to avoid problems with differences in C runtime libraries.

BfvHistoryDumpModChan

Purpose

Dumps the contents of the driver's history buffer for the specified module and channel number to the specified open file.

Syntax

```
void
BfvHistoryDumpModChan        (args)
   struct args_dh           *args;
```

The structure contains the following fields.

Input Fields

```
FILE *fp;
unsigned int module;
unsigned int channel;
int continuous_output;
int raw_dump;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.fp

FILE * pointer to an open file to receive the channel's history.

args.module

Module number of the history to dump. Usually set to 1.

args.channel

Channel number of the history to dump. Usually set to 1.

`args.continuous_output`

When set to 1, output displays continuously (similar to tail -f) until the process is killed. During the life of the process, dump history repeatedly prints all available new history information and sleeps for 1/10 sec. History entries can be lost if activity is rapid enough to exceed the capacity of the driver history buffer during the sleep period or other periods of process inactivity.

`args.raw_dump`

When set to 1, the output is an ASCII representation of the raw history data, uninterpreted.

When set to 2, the output is the raw binary history data.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Under almost all conditions, the history should be configured for 0 physical histories and 0 application histories, and the only module/channel combination that should be dumped is module 1 and channel 1.

When configured to include application histories, there are some special module meanings. Set *args.module* to 0 to use the history for the most recent application session corresponding to *args.channel* as unit/ordinal channel number. Set *args.module* to 0xFE to use the fixed application history corresponding to *args.channel* as an index value.

The history output contains additional information, including the time the history was created, driver version, and operating system platform.

The dump history output that these *BfvHistoryDump...* functions produce is separate and independent of the Bfv API debug mode output that the *BfvDebugModeSet* function enables.

See *Debugging* in *Chapter 3* of the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information.

See Also

BfvHistoryClear...

Example

```
main()
{
    struct args_dh args;
    FILE *f = fopen ("dh.log", "w");

    ...
    BT_ZERO(args);
    args.fp = f;
    args.module = 1;
    args.channel = 1;
    BfvHistoryDumpModChan(&args);
    ...
}
```

Note: When using Windows, it may be necessary to substitute "fopen" with "_dll_fopen" to avoid problems with differences in C runtime libraries.

BfvHistoryDumpUnit

Purpose

Dumps the contents of the driver's history buffer specified by the channel number to the specified open file.

Syntax

```
void
BfvHistoryDumpUnit          (args)
   struct args_dh           *args;
```

The structure contains the following fields.

Input Fields

```
FILE *fp;
int unit;
int continuous_output;
int raw_dump;
```

Output Fields

```
RES res;
```

Modified Fields

module, channel.

Input

args

Pointer to an argument structure containing input and output fields.

args.fp

FILE * pointer to an open file to receive the channel's history.

args.unit

The number of the channel to write its history.

args.continuous_output

When set to 1, output displays continuously (similar to tail -f) until the process is killed. During the life of the process, dump history repeatedly prints all available new history information and sleeps for 1/10 sec. History entries can be lost if activity is rapid enough to exceed the capacity of the driver history buffer during the sleep period or other periods of process inactivity.

`args.raw_dump`

When set to 1, the output is an ASCII representation of the raw history data, uninterpreted.

When set to 2, the output is the raw binary history data.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

Dumps the history for the destination module and channel associated with the ordinal channel specified by *unit* that only produce output if a history was created for those values. Typically, an application calls the *BfvHistoryDumpModChan* with the values of the *args.module* and *args.channel* fields set to 1 to dump the contents of the module and channel buffers.

The history output contains additional information, including the time the history was created, driver version, and operating system platform.

The dump history output that these *BfvHistoryDump*... functions produce is separate and independent of the Bfv API debug mode output that the *BfvDebugModeSet* function enables.

See *Debugging* in *Chapter 3* of the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information.

See Also

BfvHistoryClear...

Example

```
main()
{
    int unit;
    struct args_dh args;
    FILE *f = fopen ("dh.log", "w");

    ...
    BT_ZERO(args);
    args.fp = f;
    args.unit = unit;
    BfvHistoryDumpUnit(&args);
    ...
}
```

Note: When using Windows, it may be necessary to substitute "fopen" with "_dll_fopen" to avoid problems with differences in C runtime libraries.

BfvLineDumpStructure

Purpose

Dumps the contents of the BTLINE structure and configuration structures to the specified open file.

Syntax

```
void
BfvLineDumpStructure        (lp, fp)
   BTLINE                   *lp;
   FILE                     *fp;
```

Input Fields

```
int continuous_output;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure containing the structures to dump.

fp

FILE * pointer to an open file to receive the contents of the structures.

args.continuous_output

When set to 1, output displays continuously (similar to tail -f) until the process is killed. During the life of the process, dump history repeatedly prints all available new history information and sleeps for 1/10 second. History entries can be lost if activity is rapid enough to exceed the capacity of the driver history buffer during the sleep period or other periods of process inactivity.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this function to track changing states of the line and, in conjunction with *BfvHistoryDump*... and the Bfv API debug mode, to create error report logs.

Example

```
main()
{
    BTLINE *lp;
    FILE *fp = fopen("dump.log", "w");

    ...
    BfvLineDumpStructure(lp, fp);
    ...
}
```

Note: When using Windows, it may be necessary to substitute "fopen" with "_dll_fopen" to avoid problems with differences in C runtime libraries.

Macros

LINE_ERROR_INTR (*lp*)

Accesses the stored value of the data register that was generated by the last error interrupt.

Any error interrupt value is considered a fatal error by most Bfv API routines. The following are the error interrupt values:

COMMON_ERR_XMIT_UNDERRUN_DEF	0x05
COMMON_ERR_RCV_OVERRUN_DEF	0x06
COMMON_ERR_ILLEGAL_CONFIG_DEF	0x09
COMMON_ERR_DATA_HEADER_DEF	0x0C
COMMON_ERR_FIRMWARE_PANIC_DEF	0xFF
COMMON_ERR_MODEM_ERR_DEF	0x101
COMMON_ERR_ILLEGAL_VAR_VALUE_DEF	0x102
COMMON_ERR_NO_RESOURCES_DEF	0x103
COMMON_ERR_TIMEOUT_DEF	0x104
COMMON_ERR_BAD_SEQ_DEF	0x105
COMMON_ERR_NO_DATA_MOTION_DEF	0x106
COMMON_ERR_ARRAY_OVERFLOW_DEF	0x107

See also HNG_ERROR_INTERRUPT in *Volume 6, Appendix C, Hangup Codes*.

LINE_ERR_INTR_MSG (*lp*)

Returns the text message, if any, associated with the most recently reported error interrupt accessible by using LINE_ERROR_INTR.

LINE_ERR_INTR_DATA (*lp*)

Returns the data associated with the most recently reported error interrupt accessible by using LINE_ERROR_INTR. The data size can be retrieved using LINE_ERR_INTR_DATA_SIZE.

LINE_ERR_INTR_DATA_SIZE (*lp*)

Returns the data size of the data associated with the most recently reported error interrupt accessible by using `LINE_ERROR_INTR`. The data can be retrieved using `LINE_ERR_INTR_DATA`.

LINE_INTR_OVERRUN (*lp*)

Accesses the stored value of the driver's interrupt overrun status.

0 No error.

The following interrupt overrun status value is considered a fatal error by most Bfv API routines:

1 Indicates that the driver was unable to report commands back to the application program. The incoming buffer area has filled up due to the application sleeping or running slowly.

The following values are not currently applicable:

777 Indicates that a fax command overflow occurred. Although not required, we recommend that the application reset the channel before using it again.

888 Indicates that the interrupt limit, enabled with the *BfvLineInterruptLimit* function, has been exceeded. The channel must be reset before it can be used again.

999 Indicates that at least five consecutive error commands were received. When this situation occurs, the driver puts the channel into shutdown mode, and the channel must be reset before it can be used again.

See also `HNG_INTERRUPT_OVERRUN` in *Volume 6, Appendix C, Hangup Codes*.

RES Structure Parameters

The `RES` structure is used for return status indication from virtually all Bfv API functions. The `status` field indicates the category of error, and the `line_status` field indicates the specific error within a category. Together they identify the error. See *Volume 6, Appendix B, Bfv API Structures*, for more information.

Volume 2 - Bfv-Level Call Control and Call Switching

About this Volume

Volume 2, Bfv-Level Call Control and Call Switching, provides information about the following Bfv API components:

- Bfv-level Call Control functions
- Dialing Database functions
- Call Control data structures and macro

9 - Call Control Overview

This chapter introduces call control, explaining the distinction between the two levels that Dialogic supports.

It has the following sections:

- *Bfv-Level Call Control*
- *The host communicates with the Dialogic® Brooktrout® module through the Control Interface. BSMI control messages are issued by the host application (referenced as L4) to configure the board or to instruct it to perform a specific action such as make a call, clear a call, or request the status of an interface. BSMI control messages issued by the board (referenced as L3) inform the host of the status of the interface, call events, or identify an error condition.*

Call control functions enable the application to set up, initiate, connect and disconnect calls, and perform other tasks related to the telephone network. The Bfv API provides two forms of call control:

- Bfv-API Level Call Control
- BSMI-Level Call Control

Bfv-Level Call Control

An application uses call control functions to perform the process of setting up and tearing down calls on the public telephone system. The Bfv-level offers two sets of functions to use for call control:

- High-level call control functions that simplify the programming task.

  This set of functions allows the user to implement a simplified command structure to perform the call control operations. Each command in the set automatically calls the low-level functions needed to complete the call control process.

- Low-level functions that provide the application with more control of ISDN service features.

  The low-level functions allow the user more flexibility and greater control of the features available with ISDN services, although they generally require a better understanding of the ISDN standard.

For details about the high-level and low-level call control functions, see *Chapter , Bfv-Level Call Control on page 271*.

The host communicates with the Dialogic® Brooktrout® module through the Control Interface. BSMI control messages are issued by the host application (referenced as L4) to configure the board or to instruct it to perform a specific action such as make a call, clear a call, or request the status of an interface. BSMI control messages issued by the board (referenced as L3) inform the host of the status of the interface, call events, or identify an error condition.

In addition to T1/E1 ISDN call control, BSMI supports the R2 signaling protocol. Using BSMI, you can:

- Start and stop the R2 protocol on a particular timeslot on an E1 span.
- Block or unblock an idle B-channel (the ISDN channel that handles data).
- Place an outbound call.
- Answer an inbound call.
- Disconnect a call.
- Reject an incoming call.

BSMI responds to the host with notification events such as:

- Starting and stopping the R2 protocol.
- Blocking or unblocking the B-channel.
- Seizing the line for an incoming call.
- Alerting the host and then connecting a call.
- Clearing a request.
- Notifying the host when the remote end phone is ringing.
- Notifying the host when the call is disconnected at the remote end.
- Providing a protocol error or invalid command status message.

For details about BSMI-level call control functions, see *Volume 5*.

BSMI-Level Call Control

An application uses the Boston Simple Message Interface (BSMI) level of call control functions to facilitate communication directly between the Dialogic® Brooktrout® module and the T1/E1 ISDN lines. The BSMI call control functions use messages to communicate between the module and the T1/E1 lines. The collection of messages is the interface to the ISDN component of the Boston firmware and provides all the facilities for management, call control, and performance statistics monitoring. Naming conventions applied to control messages in the Bfv API's BSMI are descriptive of the functions they serve and make it easier to develop an application. When developing an application, you do not need to have a detailed knowledge of the protocol involved although a general understanding of the call model is beneficial.

10 - Bfv-Level Call Control

This chapter describes the Bfv API-level functions an application uses for call control.

It has the following sections:

- *About Bfv API-Level Call Control*
- *Bfv API High-Level Call Control Summary*
- *Bfv API Low-Level Call Control Summary*
- *Bfv API Protocol-Specific Call Control Function Summary*
- *ISDN Services Call Control Summary*
- *Call Control Configuration File*
- *Bfv-level Call Control functions listed alphabetically*

About Bfv API-Level Call Control

The Bfv API offers two sets of call control functions: high-level functions that simplify the command structure for the user and low-level functions that provide the user with greater control of ISDN service features. This chapter describes each function and how to use it effectively.

Before your application can utilize call control features, you must configure these features for your modules and their ports. You can manually configure call control by building and editing a configuration file (*callctrl.cfg*), or you can use the graphical configuration tool (Windows operating systems only) to create or edit the call control configuration file. When you have configured call control, use the *BfvLineReset* function (see *Volume 1, Chapter 2*) to load your *callctrl.cfg* file and initialize its parameters for the system.

The Bfv API call control mechanism no longer supports applications designed to use the *teleph.cfg* or *ecc.cfg* configuration files. If your application currently uses these configuration files, you must modify it to use the current call control mechanism (see *Configuring Call Control on page 33*).

Dialogic has merged the ECC library into the Boston library. You must modify your application so that it no longer attempts to link to the ECC library.

Note: Dialogic provides TruFax® boards to support the following protocols:

- Analog Loop Start
- ISDN-BRI protocol variant.

Bfv API High-Level Call Control Summary

The Bfv API high-level call control functions consist of the set shown in *Table 11* (see *Table 12 on page 274* for a list of the low-level functions). Detailed information about these high-level functions begins on *page 280* and continues on *page 362*. A description of the data structure these high-level functions use starts on *page 468*.

Table 11. High-Level Call Control Functions Summary

Function	Purpose	Page
BfvCallReject	Rejects an incoming call on line types or protocols that allow call rejection.	*293*
BfvLineAnswer	Answers an incoming call.	*362*
BfvLineCCProtocolGet	Retrieves the protocol assigned to the module.	*365*
BfvLineDialString	Places the line in an OFF_HOOK state, dials the digits specified, and returns after dialing the last digit.	*368*
BfvLineOriginateCall	Starts to divert an incoming call and waits for the process to complete on a digital line using the QSIG protocol.	*374*
BfvLineOriginateCall	Places an outgoing call.	*374*
BfvLineTerminateCall	Places the line in an ON_HOOK state.	*392*
BfvLineTransfer	Automatically transfers an incoming call from the called party to the dialed transfer number, or returns control to the application so that it can determine whether to complete or cancel the transfer. The SR140 does not support this function.	*396*
BfvLineTransferCancel	Ends a previously initiated call transfer and retrieves the original calling party. The SR140 does not support this function.	*402*
BfvLineTransferCapabilityQuery	Queries a channel's transfer capability and provides the application with information about pairs of lines available to perform two B-channel call transfers.	*404*
BfvLineTransferComplete	Completes the call transfer connection for a previously initiated call transfer. The SR140 does not support this function.	*406*
BfvLineWaitForCall	Waits for an incoming call.	*408*
BfvLoopCurrentDetectDisable	Turns off loop current detection.	*416*
BfvLoopCurrentDetectEnable	Turns on loop current detection.	*418*

Bfv API Low-Level Call Control Summary

Table 12 groups the low-level call control functions by type. See *Table 14 on page 277* for information about high-level and low-level call control functions. Detailed information about these functions begins on *page 280*. See *page 439* for descriptions of the fields in the data structures that these functions use.

Table 12. Low-Level Call Control Function Summary

Function Type	Function Name	Purpose	Page
Incoming Call	*BfvCallAccept*	Starts answering an incoming telephone call.	*280*
	BfvCallReject	Rejects an incoming telephone call.	*293*
	BfvCallRingDetect	Turns detection of ring signals on or off and determines the type of detection for notification of incoming calls.	*297*
	BfvCallSendAlerting	Sends an ALERTING message to the remote end after detecting an incoming call.	*301*
	BfvCallWaitForAccept	Finishes the process of answering an incoming telephone call.	*331*
	BfvCallWaitForSetup	Waits for an incoming call, and returns all available information about the call to the application.	*352*
Outgoing Call	*BfvCallSetup*	Starts the process of dialing an outgoing telephone call or transferring a call. The SR140 does not support call transfer.	*303*
	BfvCallWaitForAlerting	Waits for an outgoing telephone call to finish dialing or become established.	*334*
	BfvCallWaitForComplete	Waits for the outgoing telephone call to finish.	*337*
Call Disconnect	*BfvCallDisconnect*	Starts the process of terminating a telephone call.	*287*
	BfvCallWaitForRelease	Waits for the termination of a telephone call to finish.	*347*

Table 12. Low-Level Call Control Function Summary (Continued)

Function Type	Function Name	Purpose	Page
Call Diversion	*BfvCallWaitForHold*	Finishes the process of diverting an incoming call on a digital port using the QSIG protocol.	345
Call Transfer	*BfvCallHold*	Places the Bfv API in the hold state. The SR140 does not support this function.	289
	BfvCallRetrieve	Takes the Bfv API out of the hold state.The SR140 does not support this function.	295
	BfvCallTransferComplete	Completes the call transfer operation but does not wait for the transfer to finish. The SR140 does not support this function.	329
	BfvCallWaitForHold	Waits for the Bfv API to finish transitioning to the hold state. The SR140 does not support this function.	345
	BfvCallWaitForRetrieve	Waits for the Bfv API to finish transitioning out of the hold state. The SR140 does not support this function.	350
	BfvCallWaitTransferComplete	Waits for the transfer complete command to finish. The SR140 does not support this function.	359
Initialization	*BfvCallCtrlInit*	Initializes the call control runtime environment. You should use this function only for backward compatibility or to modify call control parameters directly from the Bfv API instead of changing a file. Dialogic advises using the *BfvLineReset* (see *Volume 1, Chapter 2*) function to load and initialize call control configuration parameters.	284
Shut down	*BfvCallCtrlClose*	Disables signaling on all the ISDN spans in the system, and shuts down the call control library.	283
Status	*BfvCallStatus*	Retrieves the channel's current call state.	326

Bfv API Protocol-Specific Call Control Function Summary

The functions listed in *Table 13* can only be called when using specific protocols.

Table 13. Protocol-Specific Call Control Function Summary

Function	Purpose	Page
BfvCallReconfigureHostModule	Forces the specified third party IP call control stack (host module) to read its call control configuration file again while the system is running.	*291*
BfvCallSignalingStateMonitor	Turns inbound call signaling state monitoring on or off and sets up an optional callback function.	*319*
BfvCallSignalingStateSet	Sets the outbound call signaling state to one value for a time and then to a second value. **Note:** See the description below for information about this function.	*323*

Applications can only use the *BfvCallSignalingStateMonitor* and *BfvCallSignalingStateSet* functions for T1 robbed-bit signaling (RBS) and E1 Channel Associated Signaling (CAS) protocols. The *BfvCallStatus* function provides a more generic version for the protocols to retrieve the current call state of a channel.

Do not use the **BfvCallSignalingStateSet** function with any other call control function because it provides an alternative way of controlling the module. Use of this function can cause unexpected actions to take place during the call because the function inadvertently signaled an illegal state.

ISDN Services Call Control Summary

Although the Bfv API high-level call control functions provide sufficient control for most applications, these functions do not provide a mechanism for analyzing received digits and properly accepting, rejecting, or redirecting calls for ISDN protocols.

For an application to use special features provided by ISDN services (for example, caller ID), use the low-level call control functions. *Table 14* shows the relationship between the high-level call control functions and the low-level call control functions. The high-level call control functions automatically execute the low-level functions to perform the necessary call control operations.

Table 14. Relating High- and Low-Level Call Control Functions

High-level Functions	Low-level Functions
BfvCallReject	*BfvCallDisconnect* and *BfvCallWaitForRelease*
BfvLineAnswer	*BfvCallAccept* and *BfvCallWaitForAccept*
BfvLineCCProtocolGet	No equivalent
BfvLineDialString	No equivalent
BfvLineOriginateCall	*BfvCallWaitForHold*
No equivalent	*BfvCallSendAlerting*
BfvLineOriginateCall	*BfvCallSetup*, *BfvCallWaitForAlerting*, and *BfvCallWaitForComplete*
BfvLineTerminateCall	*BfvCallDisconnect* and *BfvCallWaitForRelease*
BfvLineTransfer	*BfvCallHold*, *BfvCallSetup*, *BfvCallWaitForAlerting*, *BfvCallWaitForComplete*, and *BfvCallWaitForHold*
BfvLineTransferCancel	*BfvCallDisconnect*, *BfvCallRetrieve*, and *BfvCallWaitForRetrieve*
BfvLineTransferCapabilityQuery	No equivalent
BfvLineTransferComplete	*BfvCallTransferComplete*, *BfvCallWaitTransferComplete*, and *BfvCallWaitForRelease*
BfvLineWaitForCall	*BfvCallRingDetect* and *BfvCallWaitForSetup*
No equivalent	*BfvCallReconfigureHostModule*
BfvLoopCurrentDetectDisable	No equivalent
BfvLoopCurrentDetectEnable	No equivalent

Because the low-level functions split functionality into steps, you can perform application-specific operations between these function calls. For example: you can compare a called number against a database and redirect it to another number for call forwarding.

Event messages that formerly went to the Windows Event log now go to the Bfv API debug output. You can enable this output by calling *BfvDebugModeSet* with `DEBUG_ALL`. See *Volume 1* for more information.

Call Control Configuration File

For every installation you must create a call control configuration file that defines how you want the modules configured for the Bfv API. The sample programs provided with your Brooktrout SDK include a sample configuration file (*callctrl.cfg*) that you can edit. For Windows operating systems, Dialogic provides a graphical configuration tool that you can use to create and modify the call control configuration file. See the software installation and configuration guide that came with your software for instructions on how to use this tool.

When you have created the call control configuration file, you must modify the *call_control* parameter in your user-defined (*btcall.cfg*) configuration file to provide the file name as a null-terminated string that identifies the call control configuration file your application will use. The Bfv API configures the modules the first time your application calls the *BfvLineReset* or *BfvCallCtrlInit* function.

For detailed information about the content and set up of the user-defined configuration file and the call control configuration file, see *Volume 6, Appendix A, Configuration Files*.

BfvCallAccept

Purpose

Initiates a call answer after the *BfvCallWaitForSetup* or *BfvLineWaitForCall* function detects an incoming call.

Syntax

```
void
BfvCallAccept                (lp, args)
   BTLINE                    *lp;
   struct args_cc            *args;
```

The structure uses the following fields:

Input Field

```
char name_ident [MAX_NAME_STR];
int name_char_set;
CONNECTED_NUM connected_num;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing an output field.

args.name_ident

Specifies text identifying the name of the called party. The field allows a maximum of 50 characters (ECC_MAX_NAME_STR). Set this field only for an E1 or T1 QSIG protocol.

args.name_char_set

Specifies the international standard specification (ISOxxx) of the character set in use (only used with a QSIG protocol). Values are:

NAME_CHAR_SET_UNKNOWN −1

Unknown character set in use.

NAME_CHAR_SET_NOT_INCLUDED 0

Name does not identify a character set and the Bfv API does not send one.

`NAME_CHAR_SET_ISO8859_1` 1

Specifies use of character set defined by ISO 8859-1 international standard.

`NAME_CHAR_SET_ISO8859_2` 3

Specifies use of character set defined by ISO 8859-2 international standard.

`NAME_CHAR_SET_ISO8859_3` 4

Specifies use of character set defined by ISO 8859-3 international standard.

`NAME_CHAR_SET_ISO8859_4` 5

Specifies use of character set defined by ISO 8859-4 international standard.

`NAME_CHAR_SET_ISO8859_5` 6

Specifies use of character set defined by ISO 8859-5 international standard.

`NAME_CHAR_SET_ISO8859_7` 7

Specifies use of character set defined by ISO 8859-7 international standard.

`NAME_CHAR_SET_ISO10646_BMP` 8

Specifies use of character set defined by ISO 10646-1 and ITU-T Recommendation X.680 international standards.

`NAME_CHAR_SET_ISO10646_UTF` 9

Specifies use of character set defined by UTF-8-STRING Annex R in ISO 10646-1 international standard.

`args.connected_num`

Specifies a structure of type `CONNECTED_NUM`, containing information about the connected number that the function sends to the network as part of the CONNECT message. This field is only valid for ports using an E1 or T1 QSIG protocol. For more information, see *connected_num on page 450*.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use the *BfvCallAccept* function to initiate a call answer after using a *BfvCallWaitForSetup* or *BfvLineWaitForCall* function to detect an incoming call. To allow the channel to complete the answering process, call the *BfvCallWaitForAccept* function immediately after using *BfvCallAccept*.

BfvCallAccept starts the incoming call answering process for all protocols.

The *BfvCallAccept* function returns without waiting for the answering process to complete. For this reason, the application must call the *BfvCallWaitForAccept* function to wait and ensure that the call answering process completes before the application starts to process the incoming call. An application that fails to wait for answer completion risks starting to process an incompletely answered incoming call.

Failing to wait for the answering process to complete can prevent automatic detection of a remote hang-up in many instances. For analog lines or lines that use the T1 robbed bit protocol, this failure to wait can result in the application hearing the off-hook click or prevent automatic detection of a remote hang-up.

Your application must use this function with the *BfvCallWaitForAccept* function to start and complete the call answering process correctly for all calls on ISDN lines. If using the high-level call control functions, use the *BfvLineAnswer* function to perform the call answering process.

See Also

BfvCallRingDetect, *BfvCallWaitForAccept*, *BfvCallWaitForSetup*, *BfvLineAnswer*, *BfvLineWaitForCall*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
BfvCallRingDetect(lp, &args);
BfvCallWaitForSetup(lp, &args);

BT_ZERO(args);
BfvCallAccept(lp, &args);
```

BfvCallCtrlClose

Purpose

Shuts down the call control library.

Syntax

```
int
BfvCallCtrlClose          (void)
```

Input

None

Output

Return value:

0	Success
<1	Failed
>1	Failed

Details

Calling this function disables the signaling on all the ISDN spans in the system. The application cannot make or receive any more calls after calling this function.

BfvCallCtrlInit

Purpose

Sets up the call control runtime environment for modules.

Syntax

```
int
BfvCallCtrlInit              (args)
   struct args_cc            *args;
```

The structure uses the following fields:

Input Fields

```
char *btcall_file;
char *log_file;
int set_log_file;
```

Output Field

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.btcall_file

Pointer to a null-terminated string that identifies the full path and file name of the *btcall.cfg* user-defined configuration file. The field allows a maximum of 256 characters (MAX_PATH) and can contain spaces.

The function opens the file specified in the *args.btcall_file* argument to locate the *callctrl.cfg* file. If the *call_control* keyword does not contain a value, the call control process assumes the default value and looks for the *callctrl.cfg* file in the current working directory. See *Volume 6, Appendix A* for more information about the user-defined configuration file.

args.log_file

Specifies a null-terminated string that identifies the full path and file name of the log file. The field allows a maximum of 256 characters (MAX_PATH) and can contain spaces. If set to NULL, the system does not create a log file. The default value is NULL.

args.set_log_file

If nonzero, the value of *args.log_file* overrides the default value for the *trace_file* parameter in the call control configuration file (see *Volume 6, Appendix A*).

Output

Return value:

`<0` Error detected.

`0` Call control is in legacy T1 robbed-bit mode.

`1` Call control is in ISDN mode.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Before your application can utilize call control features, you must configure these features for your modules and their ports. You can configure call control by building or modifying a call control configuration file (*callctrl.cfg* — see *Call Control Configuration File on page 279*) or using the graphical configuration tool (Windows operating systems only).

When you have configured call control, use the *BfvLineReset* function (*see Volume 1, Chapter 2*) to load your *callctrl.cfg* file and initialize its parameters for the system. Dialogic only recommends that you use *BfvCallCtrlInit* for backward compatibility or when you need to change the configuration file.

The pointer reference to the *btcall_file* parameter is the name of your *btcall.cfg* user-defined configuration file. This file contains a pointer reference to the location of your call control configuration file.

The system initializes the call control configuration as follows:

- Applies the default configuration values. Then, any parameters found in the call control configuration file override the defaults.
- Sets the *log_file*, if the *set_log_file* parameter is set to a nonzero value.

Call *BfvCallCtrlInit* only once per process. When called, the function sets up those modules specified in the call control configuration file.

Multiple applications can access a module. Control of the module is through the first application. All other applications communicate with the module through the first application. If the first application stops, all the other applications lose the ability to make or receive calls. If this control process presents an issue for your application, You should use the BOSTON Host Service that remains active whether or not the first application stops. If you use the service, you must start it before you start any applications (see your installation and configuration guide for instructions).

Specify the call control log file name using the *args.log_file* or the *trace_file* parameter in the call control configuration file. Set *args.set_log_file* to a nonzero value. The file name can contain spaces. The name specified in the call to *BfvCallCtrlInit* has precedence over the name in the configuration file. If you do not specify a name, the system does not create a log file. If you specify a file name of `"null string"` (two double quotes without characters in between) or pass a `NULL` file name when calling *BfvCallCtrlInit*, the function disables the log file even if the configuration file had specified a name.

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

/* Setup call control using a configuration file */
BT_ZERO(args);
args.btcall_file = "btcall.cfg";
args.log_file = "call_cntrl_log.txt";
args.set_log_file = 1;
if (BfvCallCtrlInit(&args) < 0)
{
    fprintf (stderr, "Call control initialization error.\n");
    exit (1);
}
```

BfvCallDisconnect

Purpose

Starts the process of terminating a telephone call.

Syntax

```
void
BfvCallDisconnect          (lp, args)
   BTLINE                  *lp;
   struct args_cc          *args;
```

The structure uses the following fields:

Input Field

```
int cause;
int subcause;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.cause

Specifies an ISDN-defined code that provides the reason why the termination process failed. See *Volume 6, Appendix D, Defining ISDN Cause Codes*, for a description. If the underlying line protocol does not allow termination causes, the system ignores the value set in this field.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

BfvCallDisconnect initiates the process of terminating a call for all protocols.

Use the *BfvCallDisconnect* function to start the process of disconnecting a call or to clear a call that is in the connected state. You can also use the function at any time to make sure that the line is in an idle state. Use the *BfvCallWaitForRelease* function to wait for the disconnect process to complete.

Your application must use this function with the *BfvCallWaitForRelease* function to start and complete the call termination process correctly for all calls on ISDN lines. If you use the high-level call control functions, use the *BfvLineTerminateCall* function to perform the equivalent terminating process.

See Also

BfvCallWaitForRelease, *BfvLineTerminateCall*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
args.cause = IISDNcausDEFAULT;
BfvCallDisconnect(lp, &args);

BT_ZERO(args);
args.timeout = 0;
BfvCallWaitForRelease(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* It did not disconnect */
}
```

BfvCallHold

Purpose

Places the Bfv API in the hold state.

The SR140 and analog DID lines do not support this function.

Syntax

```
void
BfvCallHold                    (lp, args)
   BTLINE                      *lp;
   struct args_cc              *args;
```

The structure contains the following fields.

Input Field

None

Output Field

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing an output field.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use the *BfvCallHold* function to place the Bfv API in the hold state. This function prepares the software to handle transferring a call. Follow the *BfvCallHold* function with a call to *BfvCallWaitForHold* to wait for the Bfv API to transition to the hold state.

The application can only have one call on hold per line. To verify that the application does not already have a call on hold for the line, use the *BfvCallStatus* function to check the *args.calls_on_hold* field.

See Also

BfvCallStatus, *BfvCallWaitForHold*, *BfvCallRetrieve*, *BfvCallWaitForRetrieve*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
BfvCallHold(lp, &args);

BT_ZERO(args);
BfvCallWaitForHold(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* The API did not change to a hold state */
}
```

BfvCallReconfigureHostModule

Purpose

Forces the specified third party call control stack (host module) to read its call control configuration file again while the system is running.

Syntax

```
int
BfvCallReconfigureHostModule (
   int                        host_module_number)
```

Input Field

```
int host_module_number;
```

Output Field

None

Input

host_module_number

Identifies the number assigned to the third party IP call control stack.

The value provided for this variable ranges from 1 to 9 and must correspond to the number assigned to the call control stack in the *host_module.#* parameter in your call configuration file (see *Internet Protocol (IP) Call Control Configuration Parameters* in *Volume 6, Appendix A*).

Output

Return value:

0 Success

–1 The third party call control stack does not support reconfiguring call control parameters

>1 Failed

Details

Calling this function allows the application to force the specified call control stack to reread its configuration parameters if the stack permits reconfiguration.

At the option of the third party providing the call control stack, the stack can limit the number of reconfigurable parameters or choose not to support reconfiguration at all. When the stack only permits reconfiguring certain parameters, the details about these parameters can be found in the third party documentation that comes with your call control stack product. The third party IP call control stacks that Dialogic provides with its Brooktrout SDK support reconfiguration as follows:

- H.323 — Does not permit rereading configuration parameters.
- SIP — Permits reconfiguring all parameters except `sip_max_sessions`.

Example

```
int host_module_number = 1;
BfvCallReconfigureHostModule (host_module_number);
```

BfvCallReject

Purpose

Rejects an incoming call detected by the *BfvCallWaitForSetup* or *BfvLineWaitForCall* function.

Syntax

```
void
BfvCallReject                (lp, args)
   BTLINE                    *lp;
   struct args_cc            *args;
```

The structure contains the following fields:

Input Field

int *cause;*

Output Field

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.cause

Specifies an ISDN-defined code that provides the reason why rejecting the call failed. See *Volume 6, Appendix D, Defining ISDN Cause Codes* for a description. If the underlying line protocol does not allow termination causes, the system ignores the value set in this field.

args.subcause

Specifies the SIP response code to use in rejecting the call. This field should only be used for SIP IP protocol calls and will be ignored for all other protocols. The definitions of these response codes are specified in RFC 3261.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

BfvCallReject handles refusal of an incoming call detected by the *BfvCallWaitForSetup* or *BfvLineWaitForCall* function. *BfvCallReject* waits for the call to clear before it returns.

Not all line types and protocols allow the application to reject an incoming call. Note the following:

- On digital lines using the T1 robbed bit protocol, *BfvCallReject* blocks further processing until the incoming caller hangs up.
- On analog lines, *BfvCallReject* ignores the call but allows the call to reappear as a new call when the next ring occurs.
- On analog DID lines, the far end must clear to complete the rejection.

See Also

BfvCallDisconnect, *BfvCallWaitForRelease*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
args.timeout = 0;
args.cause = IISDNcausNUM_CHANGED;    // ISDN cause code 22 (0x16)

BfvCallWaitForSetup(lp, &args);
if (args.res.status == BT_STATUS_OK)
{
if (strncmp(args.cres.dest_id, "7814494100", 10)
{
    BT_ZERO(args);
    BfvCallReject(lp, &args);
}
else
{
    BT_ZERO(args);
    BfvCallAccept(lp, &args);
}
}
```

BfvCallRetrieve

Purpose

Takes the Bfv API out of the hold state.

The SR140 and analog DID lines do not support this function.

Syntax

```
void
BfvCallRetrieve              (lp, args)
   BTLINE                    *lp;
   struct args_cc            *args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing an output field.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this function to take the Bfv API out of the hold state. Follow this function call with a call to *BfvCallWaitForRetrieve* to wait for the transition to complete.

To verify that the application has a call on hold for the line, use the *BfvCallStatus* function to check the *args.calls_on_hold* field.

See Also

BfvCallStatus, *BfvCallWaitForRetrieve*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
BfvCallRetrieve(lp, &args);

BT_ZERO(args);
BfvCallWaitForRetrieve(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* The API did not exit the hold state */
}
```

BfvCallRingDetect

Purpose

Turns detection of ring signals on or off and determines the type of detection for notification of incoming calls.

Syntax

```
void
BfvCallRingDetect          (lp, args)
   BTLINE                  *lp;
   struct args_cc          *args;
```

The structure contains the following fields:

Input Field

```
int mode;
char *phonenum;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.mode

Specifies a value that determines how and whether the function detects incoming ring signals. Set this field as follows:

0	Turns ring detection behavior and static ring detection mode off.
–1	Turns static ring detection mode on.
1	Turns ring detection behavior on.

args.phonenum

Pointer to a null-terminated ASCII string that identifies the DID phone number the incoming call must match in order for the system to present the call to the application.

Only calls using an internet protocol (IP) such as SIP or H.323 can use this field. PSTN line types ignore the contents of this field.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The application uses this function to enable ring detection or static ring detection so that it receives notification of an incoming call. Use this function before the *BfvCallWaitForSetup* function that waits for an incoming call.

If the application uses the function to disable ring detection and the underlying call transport mechanism supports call rejection, the channel automatically rejects the incoming call without notifying the user application.

When you turn on static ring detection mode on a module's configuration (see *Volume 6, Appendix A, Configuration Files*), your application must specifically turn on the ring detection behavior on each call control channel that your application uses to receive inbound calls. Turn this behavior on by waiting for a call or calling this function with *args.mode* set to `1`. After turning on static ring detection mode and ring detection behavior on a call control channel, static ring detection mode and ring detection behavior remain turned on until specifically turned off by calling the function with *args.mode* set to `0`. If you turned static ring detection mode off on a call control channel and you want to turn it back on, you must call the *BfvCallRingDetect()* function with *args.mode* set to `–1`. Setting the *args.mode* field to –1 also turns on the ring detection behavior on the call control channel. When you turn on both static ring detection mode and ring detection behavior on a call control channel, you must expect to receive calls on that channel. Setting the *args.mode* field to a value of `1` only turns on the ring detection behavior on a call control channel.

Before Dialogic added this static ring detection feature to the configuration file, an application needed to enable ring detection after each call. The application called the *BfvCallRingDetect()* function with *args.mode* set to `1` or simply waited for an inbound

call. The Bfv API then rejected any inbound call on a channel without ring detection enabled. This behavior increased the likelihood of missing an inbound call.

Using the static ring detection feature allows an application to turn on static ring detection mode just once on a channel. When a channel has static ring detection mode enabled, the Bfv API buffers an inbound call if the channel is not in the wait for call state. After the channel enters the wait for call state, the Bfv API routes the buffered inbound call to the waiting channel. If static ring detection mode is enabled and the channel is already in the wait for call state, the Bfv API routes the inbound call directly to the channel without any buffering. This behavior increases the likelihood of answering an inbound call.

If you turn static ring detection mode on for a module, but do not enable ring detection behavior on the call control channel at least once, the Bfv API will reject an inbound call on that call control channel. If you turn static ring detection mode off for a module and a call control channel is not waiting for an inbound call, the Bfv API will reject an inbound call on that call control channel.

An application can use the *args.phonenum* input field to implement basic inbound call routing for an IP-enabled application. Unlike applications for PSTN line types, any channel in the system can answer incoming IP calls. The *args.phonenum* field allows the application to specify a DID number that an incoming call must match before the system presents the call to the application. The Bfv API only attempts to match the number of digits passed into this function in this pointer. For example, if you specify 4 digits to pass in, the Bfv API only seeks to match the first 4 digits of the incoming call, and it ignores any extra digits in the incoming call's DID string. This field only works for incoming calls using SIP and H.323 internet protocols. The Bfv API ignores the field for any incoming call on PSTN line types.

If using the high-level call control functions, use the *BfvLineWaitForCall* function to process an incoming call. The *BfvLineWaitForCall* function automatically enables ring detection.

See Also

BfvCallAccept, *BfvCallReject*, *BfvCallWaitForSetup*, *BfvLineWaitForCall*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
args.mode = 1;
   /* Use next line example only to implement call routing
      for an IP-enabled application */
args.phonenum = "4083700881";
BfvCallRingDetect(lp, &args);

BT_ZERO(args);
BfvCallWaitForSetup(lp, &args);

BT_ZERO(args);
BfvCallAccept(lp, &args);
```

BfvCallSendAlerting

Purpose

Sends an ALERTING message to the remote end after detecting an incoming call.

The SR140 does not support this function.

Syntax

```
void
BfvCallSendAlerting          (lp, args)
   BTLINE                    *lp;
   struct args_cc            *args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing an output field.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

For protocols that support call diversion, an application can use this function rather than the automatic feature to send an alerting message to the remote end after the system detects an incoming call. The function is only effective if the application turns off the automatic call alerting feature by setting the *disable_alerting* parameter to `TRUE` in the call configuration file (see *Volume 6, Appendix A, Configuration Files*). With the automatic feature turned on (*disable_alerting* = `FALSE` = default value), the Bfv API expects to send the ALERTING message automatically whenever it detects an incoming call with a diversion request.

Note: The application cannot use this function after it uses a function that answers a call.

The Bfv API does not report an error condition if the application calls this function and the Bfv API has already sent an alerting message automatically. If the protocol in use does not support this feature, the Bfv API silently ignores the request implied in the *BfvCallSendAlerting* function.

See Also

BfvCallWaitForHold, *BfvCallWaitForSetup*, *BfvLineOriginateCall*, *BfvLineWaitForCall*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
BfvCallSendAlerting(lp, &args);
```

BfvCallSetup

Purpose

Starts the process of making an outgoing telephone call.

Analog DID lines do not support this function.

Syntax

```
void
BfvCallSetup                (lp, args)
   BTLINE                   *lp;
   struct args_cc           *args;
```

The structure contains the following fields:

Input Fields

```
int call_protocol_code;
const char *calling_party;
unsigned char calling_party_presentation;
unsigned char calling_party_screening;
enum TRxECCCallType call_type;
char *phonenum;
int enquiry_call;
int ie_count;
int ie_length;
unsigned char *ie_data;
char name_ident[ECC_MAX_NAME_STR];
int name_char_set;
enum TRxTransportType call_transport;
unsigned num_user_sip_headers;
struct BT_USER_SIP_HEADER *user_sip_headers;
int fax_media_feature_tag;
unsigned fallback_rtp_reinvite;
unsigned char override_calling_plan;
unsigned char override_calling_type;
unsigned char override_called_plan;
unsigned char override_called_type;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.call_protocol_code

Specifies a calling protocol for the module that determines how and when the module reports the results of call progress analysis. Set the value of this field to one of the following (see *page 441* for descriptions):

CALL_PROTOCOL_FAX

CALL_PROTOCOL_RAW

CALL_PROTOCOL_VOICE

CALL_PROTOCOL_FAX_NO_RAW

CALL_PROTOCOL_VOICE_NO_RAW

args.calling_party

Specifies a pointer to a null-terminated ASCII string that the channel sends as the calling party information to the remote side during an outbound telephone call. Not all protocols support this feature.

If you set this field to NULL, the channel sends the caller ID from the *callctrl.cfg* file when appropriate.

args.calling_party_presentation

Specifies a value for ISDN calls that indicates the origin of the calling party number presented to the called party or the accessibility level of the calling party number. Set this field to one of the following values (see *page 442* for descriptions):

ECC_PRES_ALLOWED

ECC_PRES_RESTRICTED

ECC_PRES_NUM_NOT_AVAIL

In order to use one of the values above on a per call basis you have to set the call control presentation parameter to APP_DEFINED. Refer to the presentation parameter under the *Call Control Configuration File on page 1170*.

`args.calling_party_screening`

Specifies a value for ISDN calls that indicates the origin and validity of the calling party number passed to the called party. Set this field to one of the following values (see *page 442* for descriptions):

`ECC_SCRN_USER_NOT_SCREENED`

`ECC_SCRN_USER_VERIFICATION_PASSED`

`ECC_SCRN_USER_VERIFICATION_FAILED`

`ECC_SCRN_NETWORK_PROVIDED`

In order to use one of the values above on a per call basis you have to set the call control screening parameter to APP_DEFINED. Refer to the screening parameter under the *Call Control Configuration File on page 1170*.

`args.call_type`

Specifies the call type to use when making the outbound call. Use one of the following values for this field (see *page 443* for descriptions):

`ECC_CALL_TYPE_AUTO`

`ECC_CALL_TYPE_DEFAULT`

`ECC_CALL_TYPE_MODEM`

`ECC_CALL_TYPE_VOICE`

Note: Not all protocols support this feature. If the protocol does not support the feature, the system ignores the *args.call_type* field.

`args.phonenum`

Specifies a pointer to a null-terminated ASCII string that identifies the phone number to dial. Dial strings (phone numbers) may be up to a maximum of 255 characters (ECC_MAX_DIGIT_STR - 1) for most protocols and locations. See Details.

PSTN Telephony

The dial string field supports the following digits and control characters. Invalid characters are ignored; upper and lower case letters are equivalent. Some protocols ignore control characters and only accept DTMF characters.

Valid Digits and Control Characters

0 - 9	Dials digits ‘0’ through ‘9’.
# (pound)	Dials a pound.
* (asterisk)	Dials an asterisk.
A - D	Sends the DTMF tone corresponding to the specified alphabetic character.
p	Changes the current or default dialing mode from tone dialing to pulse dialing.
t	Changes the current or default dialing mode from pulse dialing to tone dialing.
w	Waits for dial tone.
, (comma)	Causes a 1-second pause.
; (semicolon), i or I	Causes a 5-second pause. To create longer pauses, string any of these characters together.
! (exclamation point)	Sends a hook flash on analog and T1 robbed bit modules.

Note: In an analog environment or when using a T1 robbed bit FXS loopstart or E1 CAS loopstart protocol, the ‘w’ character means wait for dial tone. All other protocols ignore the ‘w’ and ‘i’ characters. Only analog environments and T1 robbed bit or E1 CAS protocols use the ‘p’, ‘t’, comma and semicolon characters.

IP Telephony

For IP outbound calls using the H.323 protocol:

- `Phone#@IP Address:Port#`

  If the receiving side does not require a phone number, a value for `Phone#` is optional. Also, `:Port#` is optional — the Bfv API uses 1720 as the default port value.

- `TA:IP Address:Port#,Phone#`

  `:Port#` is optional — the Bfv API uses 1720 as the default port value. If the receiving side does not require a phone number, a value for `Phone#` is optional.

- `Name:<name of person to dial>`

  Use this form only if using gatekeeper

- `E164alias:7894561234`

  Use this form only if using gatekeeper

Note: DNS lookups are not supported in H.323. You can use an H.323 or E.164 alias in conjunction with a gatekeeper to provide similar functionality.

Examples

```
4082345555@10.155.89.6:175
4082345555@10.155.89.6

TA:10.155.89.6:175,4082345555
TA:10.155.89.6,4082345555
TA:10.155.89.6

Name:Fred Smith
E164alias:4082345555
```

For IP outbound calls using the SIP protocol:

```
Phone#@IP Address:Port#
```

If the receiving side does not require a phone number, a value for Phone# is optional. Also, :Port# is optional - the Bfv API uses 5060 as the default port value.

Examples (IPv4 Addresses)

```
4082345555@10.155.89.7:175
4082345555@10.155.89.7
```

Examples (IPv6 Addresses)

```
4082345555@[2000::2ef3:1dff:ea3]:175
4082345555@[fe80::1f4:189c:74da:69f7]
```

Note: IPv6 addresses must be enclosed in brackets. In addition, if a link-local IPv6 address is specified, the Scope ID should be omitted from the address.

DTMF Post Dialing

For all types of IP calls, the character '&' (ampersand) may be included to initiate post-dialing. This character indicates that the rest of phonenum specifies a sequence of DTMF digits to be "post-dialed" after messages from the remote side indicate the call is proceeding towards connecting.

Within the post-dial string, all dialing characters listed for PSTN Telephony are allowed except for 'p', 't', 'w', and '!'. The appearance of an additional '&' will terminate processing of the string.

Post-dialing of the specified digits will occur upon the first receipt of one of the following IP call control messages:

- SIP -- 183 Progress
- SIP -- 200 OK
- H.323 -- Progress
- H.323 -- Connect

The post-dial feature is controlled by the user configuration file parameter *post_dialing_enable*. If the feature is disabled by that parameter, then the ampersand has no special effect and the entire phonenum field is used as is.

Note: If a program accepts a phone number on the command line, the phone number will likely need to be quoted if it contains an ampersand, since this is a special character on most OSes. (Use double quotes (") on Windows, or single (') or double (") quotes on Linux.)

`args.enquiry_call`

Specifies a Boolean value that indicates whether the function sets up the outbound call to handle it as a transfer from the called party. Set the field as follows:

`FALSE`

Indicates that the outbound call is not set up to make a transfer.

`TRUE`

Indicates that the outbound call expects to make a transfer from the called party.

When the application sets the value in this field to `TRUE` and the line only supports single B-channel call transfer, the application must put the line on hold first using the *BfvCallHold* function. The application does not need to use the *BfvCallHold* function when the line support provides two B-channels for the transfer.

`args.ie_count`

Specifies the number of custom information elements (IE) to send.

`args.ie_length`

Specifies the number of bytes that the custom *args.ie_data* field contains.

`args.ie_data`

Specifies an array of hexadecimal characters that indicate the content of the custom IE.

args.name_ident

Specifies text identifying the name of the called party. The field allows a maximum of 50 characters (`ECC_MAX_NAME_STR`). Set this field only for an E1 or T1 QSIG protocol.

args.name_char_set

Specifies the international standard specification (ISOxxx) of the character set in use (only used with a QSIG protocol). Values are:

`NAME_CHAR_SET_UNKNOWN` `-1`

Unknown character set in use.

`NAME_CHAR_SET_NOT_INCLUDED` `0`

Name does not identify a character set and the Bfv API does not send one.

`NAME_CHAR_SET_ISO8859_1` `1`

Specifies use of character set defined by ISO 8859-1 international standard.

`NAME_CHAR_SET_ISO8859_2` `3`

Specifies use of character set defined by ISO 8859-2 international standard.

`NAME_CHAR_SET_ISO8859_3` `4`

Specifies use of character set defined by ISO 8859-3 international standard.

`NAME_CHAR_SET_ISO8859_4` `5`

Specifies use of character set defined by ISO 8859-4 international standard.

`NAME_CHAR_SET_ISO8859_5` `6`

Specifies use of character set defined by ISO 8859-5 international standard.

`NAME_CHAR_SET_ISO8859_7` `7`

Specifies use of character set defined by ISO 8859-7 international standard.

`NAME_CHAR_SET_ISO10646_BMP` `8`

Specifies use of character set defined by ISO 10646-1 and ITU-T Recommendation X.680 international standards.

`NAME_CHAR_SET_ISO10646_UTF` `9`

Specifies use of character set defined by UTF-8-STRING Annex R in ISO 10646-1 international standard.

`args.call_transport`

Selects the transport protocol to use for an outbound SIP call from one of the following:

TRANSPORT_TYPE_UDP

Select User Datagram Protocol (UDP) as the transport protocol to use for the outbound SIP call.

TRANSPORT_TYPE_TCP

Select Transmission Control Protocol (TCP) as the transport protocol to use for the outbound SIP call. In order to use this setting, TCP protocol support must be enabled in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

TRANSPORT_TYPE_DEFAULT

Use the default call transport, either UDP or TCP for the outbound SIP call. If the default call transport is not explicitly specified in the call control configuration file, UDP will be used. Otherwise, the transport protocol used for the outbound SIP call will be what's specified in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

NOTE: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

`args.num_user_sip_headers`

Specifies the number of entries in the array of *BT_USER_SIP_HEADER* structures referenced by the *args.user_sip_headers* field. The maximum number of entries in the *BT_USER_SIP_HEADER* structure array is *BT_USER_SIP_HEADERS_MAX_NUM_HEADERS*.

NOTE: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

args.user_sip_headers

Specifies a reference to an array of *BT_USER_SIP_HEADER* structures that specify SIP header names and values to add to the initial SIP INVITE of an outbound call. The *BT_USER_SIP_HEADER* structure is defined below:

```
struct BT_USER_SIP_HEADER {
    char *header_name;
    char *header_value;
};
```

The number of entries in the array of *BT_USER_SIP_HEADER* structures referenced by the *args.user_sip_headers* field is specified by the value of the *num_user_sip_headers* field and can have a maximum size of *BT_USER_SIP_HEADERS_MAX_NUM_HEADERS*. If the value specified by *num_user_sip_headers* is 0, this field is ignored.

The *header_name* field in the *BT_USER_SIP_HEADER* structure points to a null-terminated ASCII string that identifies the name of the SIP header to add to the initial SIP INVITE for an outbound SIP call. The maximum length that can be specified for the header name (including the null terminator) is *BT_USER_SIP_HEADERS_MAX_NAME_LEN*.

The *header_value* field in the *BT_USER_SIP_HEADER* structure points to a null-terminated ASCII string that identifies the value of the SIP header to add to the initial SIP INVITE for an outbound SIP call. The maximum length that can be specified for the header value (including the null terminator) is *BT_USER_SIP_HEADERS_MAX_VALUE_LEN*.

Applications should not attempt to specify standard SIP headers (e.g. To, From, Via, Call-ID, CSeq, Contact, etc.) using the *args.user_sip_headers* field as this may result in unpredictable and unsupported behavior.

NOTE: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

args.fax_media_feature_tag

Specifies a "sip.fax" media feature tag value to add to an Accept-Contact header in the outbound SIP INVITE request. Set this field to one of the following values:

BT_FAX_MEDIA_FEATURE_TAG_DEFAULT

Set "sip.fax" media feature tag to a default value based on the *fax_transport_protocol* parameter value in the *t38parameters* section of the *callctrl.cfg* file as specified in the following table:

fax_transport_protocol value	"sip.fax" value
t38_never	passthrough
t38_only	t38
t38_first	t38
Not specified in *callctrl.cfg* file	t38

BT_FAX_MEDIA_FEATURE_TAG_T38

Set" sip.fax" media feature tag to "t38".

BT_FAX_MEDIA_FEATURE_TAG_PASSTHROUGH

Set "sip.fax" media feature tag to "passthrough".

BT_FAX_MEDIA_FEATURE_TAG_DISABLED

Do not add "sip.fax" media feature tag to transmitted SIP INVITE message.

If a *fax_media_feature_tag* value is specified that is not supported by the current configuration (e.g., *BT_FAX_MEDIA_FEATURE_TAG_T38* specified but *fax_transport_protocol* value set to *t38_never* in the *callctrl.cfg* configuration file), the outbound SIP call will fail with the *status* field in the RES results structure set to *BT_STATUS_ERROR* and the *line_status* field in the RES results structure set to *APIERR_CALL_CONTROL*.

In order to use this field for outbound SIP calls, RFC 6913 feature support must be enabled in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

`args.fallback_rtp_reinvite`

Specifies whether or not a SIP RTP reINVITE should be transmitted for G.711 fallback mode if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable). Valid values are:

BT_FALLBACK_RTP_REINVITE_DEFAULT

Default to the setting specified in the call control configuration file.

BT_FALLBACK_RTP_REINVITE_DISABLE

Do not transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

BT_FALLBACK_RTP_REINVITE_ENABLE

Transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

When the application sets the value in this field to *BT_FALLBACK_RTP_REINVITE_DEFAULT*, the functionality will default to the setting specified in the call control configuration file (*callctrl.cfg*) by the *g711_fallback_rtp_reinvite* parameter. If the *g711_fallback_rtp_reinvite* parameter isn't specified in the call control configuration file, then the default functionality is *BT_FALLBACK_RTP_REINVITE_DISABLE*.

An application can override the value specified in the call control configuration file by the *g711_fallback_rtp_reinvite* parameter by setting this field to a value of either *BT_FALLBACK_RTP_REINVITE_DISABLE* or *BT_FALLBACK_RTP_REINVITE_ENABLE*.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_DISABLE* will prevent transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_ENABLE* will result in transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable) and the fax transport protocol (*fax_transport_protocol*) parameter specified in the call control configuration file is set to *t38_first*. The SDP settings in the SIP RTP reINVITE will be the same RTP codec settings initially used to establish the call.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol or PSTN line types.

`args.override_calling_plan`

Specifies the calling party telephone numbering plan used for outbound calls. Set this field to one of the following values (see *page 451* for descriptions):

`ECC_NUM_PLAN_UNKNOWN`

`ECC_NUM_PLAN_ISDN`

`ECC_NUM_PLAN_TELEPHONY`

`ECC_NUM_PLAN_PRIVATE`

The value specified will only be active if it is ORed with the symbol *OVERRIDE_NUMBERING_FLAG*. For example:

`ECC_NUM_PLAN_ISDN | OVERRIDE_NUMBERING_FLAG`

If it is not active, the value specified by the *numbering_plan* parameter will be used. Refer to the *numbering_plan* parameter under the *Call Control Configuration File on page 1170*.

`args.override_calling_type`

Specifies the type of calling party telephone number used for outbound calls. Set this field to one of the following values (see *page 451* for descriptions):

`ECC_NUM_TYPE_UNKNOWN`

`ECC_NUM_TYPE_INTERNATIONAL`

`ECC_NUM_TYPE_NATIONAL`

`ECC_NUM_TYPE_SUBSCRIBER`

`ECC_NUM_TYPE_ABBREVIATED`

The value specified will only be active if it is ORed with the symbol *OVERRIDE_NUMBERING_FLAG*. For example:

`ECC_NUM_TYPE_SUBSCRIBER | OVERRIDE_NUMBERING_FLAG`

If it is not active, the value specified by the *numbering_type* parameter will be used. Refer to the *numbering_type* parameter under the *Call Control Configuration File on page 1170*.

`args.override_called_plan`

Specifies the called party telephone numbering plan used for outbound calls. Set this field to one of the following values (see *page 452* for descriptions):

`ECC_NUM_PLAN_UNKNOWN`

`ECC_NUM_PLAN_ISDN`

`ECC_NUM_PLAN_TELEPHONY`

`ECC_NUM_PLAN_PRIVATE`

The value specified will only be active if it is ORed with the symbol *OVERRIDE_NUMBERING_FLAG*. For example:

`ECC_NUM_PLAN_ISDN | OVERRIDE_NUMBERING_FLAG`

If it is not active, the value specified by the *numbering_plan* parameter will be used. Refer to the *numbering_plan* parameter under the *Call Control Configuration File on page 1170*.

`args.override_called_type`

Specifies the type of called party telephone number used for outbound calls. Set this field to one of the following values (see *page 452* for descriptions):

`ECC_NUM_TYPE_UNKNOWN`

`ECC_NUM_TYPE_INTERNATIONAL`

`ECC_NUM_TYPE_NATIONAL`

`ECC_NUM_TYPE_SUBSCRIBER`

`ECC_NUM_TYPE_ABBREVIATED`

The value specified will only be active if it is ORed with the symbol *OVERRIDE_NUMBERING_FLAG*. For example:

`ECC_NUM_TYPE_SUBSCRIBER | OVERRIDE_NUMBERING_FLAG`

If it is not active, the value specified by the *numbering_type* parameter will be used. Refer to the *numbering_type* parameter under the *Call Control Configuration File on page 1170*.

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

BfvCallSetup initiates an outbound call or a call transfer. Follow this function call with a call to *BfvCallWaitForComplete* that waits for the outbound calling process to finish establishing the call.

Note: The SR140 does not support call transfer. Analog DID lines do not support outbound calling.

For applications that want to take the channel off hook and make it busy whether or not any call is coming in, use this function and dial a comma (pause character), but do not complete the process by following this function call with a call to *BfvCallWaitForComplete*. When you are ready to put the channel on hook again, call the *BfvLineTerminateCall* function.

To initiate the outbound call as a transfer, set *args.enquiry_call* to `TRUE`. If the line supports two B-channel transfers, *BfvCallSetup* starts a transfer process and the application uses *BfvCallWaitForComplete* to wait for the outbound calling process to finish establishing the call. Use the *BfvCallTransferComplete* function to complete the transfer.

If the application needs to cancel the transfer, use the *BfvCallDisconnect* function to start the release of the enquiry call.

If the line only supports single channel transfers, the application must first put the line on hold using *BfvCallHold* and *BfvCallWaitForHold* before the *BfvCallSetup* function can start making the enquiry call. Use *BfvCallWaitForComplete* to wait for the outbound transfer to complete.

For applications using Euro-ISDN on E1 or BRI lines (but not T1 ISDN lines), this function automatically sends a dial string using the protocol's overlapped dialing feature when the phone number exceeds 20 digits. The Euro-ISDN protocol only allows applications to send 20 digits in a block when placing a call. For phone numbers exceeding 20 digits, the protocol uses a process called overlapped dialing. This process sends extra digits after the initial call setup, allowing:

- The application to dial very large phone numbers
- The remote end to start answering a call before it receives all the digits

Users can place a call with up to 255 digits in the dial string. The Bfv API automatically breaks up the dial string into multiple blocks of 20 digits, and uses the overlapped dialing feature in the protocol to send one block of digits at a time. This process does not require any changes in the application.

Some protocols or locations do not support overlapped dialing. For example, T1 ISDN only allows a maximum of 24 digits.

See Also

BfvCallDisconnect, *BfvCallHold*, *BfvCallWaitForAlerting*, *BfvCallWaitForComplete*, *BfvCallWaitForHold*, *BfvCallWaitForRelease*

Example

```
BTLINE *lp;
.
.
.
unsigned char ie_data[] = {
   0x1c, 0x12, 0x91, 0xa1, 0x0f, 0x02, 0x02, 0x00,
   0x80, 0x02, 0x01, 0x0f, 0x30, 0x06, 0x02, 0x01,
   0x05, 0x0a, 0x01, 0x01};

struct args_cc args;

BT_ZERO(args);
args.call_protocol_code = CALL_PROTOCOL_FAX;
args.phonenum = ”18005551212”;
args.ie_count = 1;
args.ie_length = sizeof(ie_data);
args.ie_data = ie_data

BfvCallSetup(lp, &args);

BT_ZERO(args);
args.call_protocol_code = CALL_PROTOCOL_FAX;
args.timeout = 0;
BfvCallWaitForComplete(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* It did not connect */
}
```

BfvCallSignalingStateMonitor

Purpose

Turns inbound call signaling state monitoring on or off and sets up an optional callback function.

The SR140 does not support this function.

Syntax

```
int
BfvCallSignalingStateMonitor(lp, args)
   BTLINE                    *lp;
   struct args_signaling     *args;
```

The structure contains the following fields.

Input Fields

```
unsigned unit;
unsigned stream;
unsigned time_slot;
void (*func)(BTLINE *lp,
      struct args_signaling *args);
unsigned mon_mode;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.unit

Specifies the telephony port unit number (see the parameters for the *Call Control Configuration File* in *Volume 6, Appendix A*).

args.stream

Specifies the stream number.

args.time_slot

Specifies the time slot number.

`args.func`

Specifies a pointer to an optional callback function. If `NULL`, no callback will be performed.

Call the function as indicated by its definition in the *Input Fields* paragraph. *Args* is a pointer to an *args_signaling* structure where the following fields contain values to describe the new signaling state:

unit	Telephony port unit number.
stream	Stream number.
time_slot	Time slot number.
stateIn_0	The signaling state of the previous inbound call.
durationIn_0	The time, in milliseconds, that the signaling state remained in effect for the previous inbound call.
stateIn_1	The new signaling state for the inbound call.
stateOut_0	The current signaling state of the outbound call. The structure and its contents must not be modified.

`args.mon_mode`

Specifies the monitoring mode. Valid values are:

`BT_MON_MODE_ENABLE`

Enables monitoring.

`BT_MON_MODE_DISABLE`

Disables monitoring.

`BT_MON_MODE_ONETIME`

Performs a one-time state check.

Output

Return value: None. This function does not have a return value. Use the return value from *BfvCallSignalingStateSet*.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Call signaling states are values whose bit representation takes the form xxxx...xxxABCD, where x indicates values to be ignored (should be 0), and A, B, C, D represent signaling bits.

Enabling call signaling state monitoring with no callback function is useful for the purpose of viewing the resulting notifications in Bfv API debug mode. Your application can only set up one callback function per module. All calls made with a non-`NULL` *args.func* value should contain the same value.

The function also provides the ability to do a one-time state check.

Note: The *BfvCallStatus* function works for all protocols and you should use it to determine the current call state.

You can call the *BfvCallSignalingStateMonitor* function only for T1 robbed-bit signaling (RBS) and E1 Channel Associated Signaling (CAS) telephony units. For all other protocols, you must use *BfvCallStatus* to determine the channel's current call state.

The callback function will be called when monitoring is first enabled and when an inbound signaling state change occurs.

To retrieve the signaling state values when performing a one-time state check, you must still supply the callback function.

For notifications to process properly and the supplied callback function to be invoked, use Bfv API functions involving the same line pointer (*lp*). If there are no other Bfv API operations required, use *BfvRcvProcessPkt* (see *Volume 1, Chapter 2*).

See Also

BfvCallSignalingStateSet, *BfvCallStatus*

Example

```
void sig_func(lp,args)
BTLINE *lp;
struct args_signaling *args;
{
   printf(
        "State info: unit %d, stream %d, time slot %d,\n"
        args->unit,args->stream,args->time_slot);
   printf("\tprev in state %x, prev time %u,\n"
        args->stateIn_0,args->durationIn_0);
   printf("\tnew in state %x, cur out state %x\n",
        args->stateIn_1,args->stateOut_0);
}

main()
{
   BTLINE *lp;
   struct args_signaling args;
   struct args_telephone args_tel;

   ...
   /* Set up signal monitoring on unit 1 */
   BT_ZERO(args);
   args.unit = 1;
   args.stream = 6;
   args.time_slot = 0;
   args.func = sig_func;
   args.mon_mode = BT_MON_MODE_ENABLE;
   BfvCallSignalingStateMonitor(lp,&args);

   BT_ZERO(args_tel);
   BfvLineWaitForCall(lp,&args_tel);
   ...
}
```

BfvCallSignalingStateSet

Purpose

Sets the signaling state for the outbound call to one value for a time and then to a second value.

The SR140 does not support this function.

Syntax

```
int
BfvCallSignalingStateSet   (lp, args)
   BTLINE                  *lp;
   struct args_signaling   *args;
```

The structure contains the following fields.

Input Fields

```
unsigned unit;
unsigned stream;
unsigned time_slot;
unsigned stateOut_0;
unsigned durationOut_0;
unsigned stateOut_1;
unsigned millisecs;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.unit

Specifies the telephony port unit number (see the *Call Control Configuration Files in Volume 6, Appendix A* parameters).

args.stream

Specifies the stream number.

`args.time_slot`

Specifies the time slot number.

`args.stateOut_0`

Specifies the first signaling state to set for the outbound call.

`args.durationOut_0`

Specifies the time, in milliseconds, to hold the first signaling state for the outbound call.

`args.stateOut_1`

Specifies the second signaling state to set for the outbound call.

Output

Return value:

`0` Success.

`>0` Operational error reported by firmware.

`<0` Other error.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Call signaling states are values whose bit representation is of the form xxxx...xxxABCD, where x are values to be ignored (should be 0), and A, B, C, D represent signaling bits. To set an absolute steady state, set *args.stateOut_0* to equal *args.stateOut_1* and set *args.durationOut_0* to 0.

Note: Call this function only for T1 robbed-bit signaling (RBS) and E1 Channel Associated Signaling (CAS) telephony units. The function does not work for any other protocol.

If you call this function for the applicable protocol, set the *set_api* parameter to the `BSMI` value in the call configuration file (see *Volume 6, Appendix A*).

Do not use this function with any other call control function because it provides an alternative way of controlling the module. Use of this function can cause unexpected actions to take place during the call because the function inadvertently signaled an illegal state.

See Also

BfvCallSignalingStateMonitor

Example

```
BTLINE *lp;
struct args_signaling args;

/* Set state on unit 1 */
BT_ZERO(args);
args.unit = 1;
args.stream = 6;
args.time_slot = 0;
args.stateOut_0 = 0x0f;
args.durationOut_0 = 1000;
args.stateOut_1 = 0x00;
args.millisecs = 1500;
BfvCallSignalingStateSet(lp,&args);
```

BfvCallStatus

Purpose

Retrieves the channel's current call state.

Syntax

```
void
BfvCallStatus                (lp, args)
   BTLINE                    *lp;
   struct args_cc            *args;
```

The structure contains the following fields:

Input Field

None

Output Fields

```
RES res;
int state;
int calls_on_hold;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing output fields for the retrieved information.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

`args.state`

Returns values that include the following (see *page 446* for definitions):

```
BST_ALERTING
BST_CALL_DETECTED
BST_CLEAR_CALL
BST_CONNECTED
BST_DIAL
BST_DIAL_COMPLETE
BST_DOWN
BST_IDLE
BST_WAIT_FOR_CALL
```

`args.calls_on_hold`

Returns the number of calls that the channel has on hold.

The TR1034 and SR140 do not support this field.

Details

Use *BfvCallStatus* to retrieve the current call state. Your application achieves the most benefit when it calls this function to detect a call state change after a call is connected.

Note: This function was previously named *BfvISDNStatus*.

Example

```
BTLINE *lp;
struct args_cc args;
struct args_line_admin args_admin;
BTERR bterr;

BT_ZERO(args);
args.unit = unit;
if ((lp = BfvLineAttach (&args_admin)) == NULL)
{
    BfvErrorMessage(lp, &args.res, &bterr);
    fprintf (stderr,"BfvLineAttach: %s\n", bterr.long_msg);
    exit (1);
}

BT_ZERO(args_admin);
args_admin.config_file_name = "btcall.cfg";
if (BfvLineReset (lp, &args_admin) < 0)
{
    BfvErrorMessage(lp, &args_admin.res, &bterr);
    fprintf (stderr, "BfvLineReset: %s: status %lX.\n",
    bterr.long_msg, args_admin.reset_status);
    exit (1);
}

BT_ZERO(args);
args.phonenum = “12345”;
BfvCallSetup(lp, &args);

BT_ZERO(args);
BfvCallWaitForComplete(lp, &args);

/* Application processing */

/* Check for Disconnect */
BT_ZERO(args);
BfvCallStatus(lp,&args);
if (args.res.status == BT_STATUS_OK && args.state == BST_CONNECTED)
{
    BT_ZERO(args);
    BfvCallDisconnect(lp,&args);

    BT_ZERO(args);
    BfvCallWaitForRelease(lp,&args);
}
```

BfvCallTransferComplete

Purpose

Completes a call transfer without waiting for the transfer process to complete.

The SR140 and analog DID lines do not support this function.

Syntax

```
void
BfvCallTransferComplete     (lp, args)
   BTLINE                   *lp;
   struct args_cc           *args;
```

The structure contains the following fields.

Input Field

BTLINE **lp_second_channel;*

Output Field

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.lp_second_channel

A reference to the *lp* of the second channel for the transfer process to use for protocols that require two B-channels to transfer a call (for example, Release Link Trunk (RLT) protocol).

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This low-level function starts the completion process of a transfer but does not wait until the transfer completes. The application must use the *BfvCallWaitTransferComplete* function to wait for the transfer complete process to finish and then use *BfvCallWaitForRelease* to wait for the system to complete the entire call transfer process.

If your application uses the high-level call control functions, use the *BfvLineTransferComplete* function to perform the equivalent completion process.

See Also

BfvCallWaitForRelease, *BfvCallWaitTransferComplete*, *BfvLineTransferComplete*

Example

```
BTLINE *lp;
BTLINE *lp_2nd;
.
.
.
struct args_cc args;
.
. /* Transfer Set up */
.
BT_ZERO(args);
args.lp_second_channel = lp_2nd;
BfvCallTransferComplete(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    BT_ZERO(args);
    BfvCallDisconnect(lp,&args);
}
else
    BT_ZERO(args);
    BfvCallWaitTransferComplete (lp,&args);
}

BT_ZERO(args);
BfvCallWaitForRelease(lp, &args);
```

BfvCallWaitForAccept

Purpose

Finishes the process of answering an incoming telephone call.

Syntax

```
void
BfvCallWaitForAccept        (lp, args)
   BTLINE                   *lp;
   struct args_cc           *args;
```

The structure contains the following fields:

Input Field

```
int timeout;
```

Output Fields

```
int cause;
int cause_location;
int subcause;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.timeout

Specifies an integer value that determines the length of time to wait for completion of an answer to the call. Valid values are:

0	Indicates waiting forever with no timeout.
nonzero	Indicates the number of milliseconds to wait for the function to complete.

Output

Return value: None.

`args.cause`

Returns an ISDN-defined code if the call fails that provides the reason why the function failed to finish answering the call. See *Volume 6, Appendix D, Defining ISDN Cause Codes* for a description.

`args.subcause`

If the call fails, returns the response code of the SIP message that resulted in the failure of the incoming call. This field is only applicable to calls using the SIP protocol. The definitions of these response codes are specified in RFC 3261.

`args.cause_location`

Returns an ISDN-defined code that indicates the originator (local or remote) of the failure notification (*args.cause*). Valid values are:

`IISDNlocUSER`	User
`IISDNlocPVT_LOCAL`	Private network serving the local user
`IISDNlocPUB_LOCAL`	Public network serving the local user
`IISDNlocTRANSIT_NET`	Transit network
`IISDNlocPUB_REMOTE`	Public network serving the remote user
`IISDNlocPVT_REMOTE`	Private network serving the remote user
`IISDNlocINTERNATIONAL`	International network
`IISDNlocBEY_INTERWORK`	Network beyond internet-working point

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use *BfvCallWaitForAccept* after calling *BfvCallAccept* to allow the channel to finish the answering process for the call.

An application that fails to wait for answer completion risks starting to process an incompletely answered incoming call.

Failing to wait for the answering process to complete can prevent automatic detection of a remote hang-up in many instances. For analog lines or lines that use the T1 robbed bit protocol, this failure to wait can result in the application hearing the off-hook click.

Your application must use this function with the *BfvCallAccept* function to start and complete the call answering process correctly for all calls. If using the high-level call control functions, use the *BfvLineAnswer* function to perform the call answering process.

See Also

BfvCallAccept, *BfvLineAnswer*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
BfvCallAccept(lp, &args);

BT_ZERO(args);
args.timeout = 0;
BfvCallWaitForAccept(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* It did not answer */
}
```

BfvCallWaitForAlerting

Purpose

Waits for an outgoing telephone call to finish establishing or dialing without waiting for the call to connect.

Analog DID lines do not support this function.

Syntax

```
int
BfvCallWaitForAlerting      (lp, args)
   BTLINE                   *lp;
   struct args_cc           *args;
```

The structure contains the following fields.

Input Fields

```
long timeout;
```

Output Fields

```
int cause;
int cause_location;
int subcause;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.timeout

Specifies an integer value that determines the length of time to wait for dialing to complete. Valid values are:

0	Indicates waiting forever with no timeout.
nonzero	Indicates the number of milliseconds to wait for the function to complete.

Output

Return value:

`0`	Successful return
`nonzero`	Error

`args.cause`

Returns an ISDN-defined code if the call fails that provides the reason why the function failed to complete the outbound call. See *Volume 6, Appendix D, Defining ISDN Cause Codes* for a description.

`args.subcause`

If the call fails, returns the response code of the SIP message that resulted in the failure of the outbound call. This field is only applicable to calls using the SIP protocol. The definitions of these response codes are specified in RFC 3261.

`args.cause_location`

Returns an ISDN-defined code that indicates the originator (local or remote) of the failure notification (*args.cause*). Valid values are:

`IISDNlocUSER`	User
`IISDNlocPVT_LOCAL`	Private network serving the local user
`IISDNlocPUB_LOCAL`	Public network serving the local user
`IISDNlocTRANSIT_NET`	Transit network
`IISDNlocPUB_REMOTE`	Public network serving the remote user
`IISDNlocPVT_REMOTE`	Private network serving the remote user
`IISDNlocINTERNATIONAL`	International network
`IISDNlocBEY_INTERWORK`	Network beyond internet-working point

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

BfvCallWaitForAlerting waits for an outbound call initiated by using *BfvCallSetup* to finish dialing or become established. You must use *BfvCallSetup* first to start the outbound call process.

BfvCallWaitForAlerting, unlike *BfvCallWaitForComplete*, does not use in-band call progress analysis. However, if the protocol provides an out-of-band indication of call answer, this indication can cause *BfvCallWaitForAlerting* to succeed for an answered call.

BfvCallWaitForAlerting can set the line state to either OFF_HOOK or CONNECTED. The application program must examine *args.res* to determine how to proceed. For example, if the call setup succeeded, the application can call *BfvCallTransferComplete* to complete the blind transfer after the dialing finishes. Note that some protocols require in-band call progress to determine if the call state is busy, ringing, or answering. If the application needs to verify one of these states before transferring the call, use *BfvCallWaitForComplete* to determine the call state.

See Also

BfvCallDisconnect, *BfvCallSetup*, *BfvCallWaitForComplete*, *BfvCallWaitForRelease*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
args.phonenum = ”13115552368”;
BfvCallSetup(lp, &args);

BT_ZERO(args);
args.timeout = 0;
BfvCallWaitForAlerting(lp, &args);
if (args.res.status != BT_STATUS_OK)
{

    /* It did not connect or go off hook*/

}
```

BfvCallWaitForComplete

Purpose

Finishes the process of establishing an outgoing telephone call by waiting for an answer to the call.

Analog DID lines do not support this function.

Syntax

```
int
BfvCallWaitForComplete      (lp, args)
   BTLINE                   *lp;
   struct args_cc           *args;
```

The structure contains the following fields.

Input Fields

```
char *arg;
int call_mode;
int call_protocol_code;
int (*func)(BTLINE *lp, char *arg);
long timeout;
unsigned sip_header_list_len;
BT_SIP_HEADER_LIST *sip_header_list;
```

Output Fields

```
int cause;
int subcause;
int cause_location;
RES res;
CALL_RES cres.name_ident;
CALL_RES cres.name_char_set;
CALL_RES cres.connected_num;
enum TRxTransportType call_transport;
unsigned sip_header_list_len;
BT_SIP_HEADER_LIST *sip_header_list;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

`args.arg`

An argument for the *func* feature. This field can be set to `NULL`.

When processing SIP 183 Session Progress response headers, the callback function must have access to the *sip_header_list* information. This is done by setting *args.arg* as follows:

```
args.arg = (char *)&args;
```

`args.call_mode`

Specifies a Boolean value that indicates whether the outbound call expects to receive a fax transmission when the connection completes. Set the field as follows:

`FALSE`

Indicates that the outbound call does not expect to receive a fax.

`TRUE`

Indicates that the outbound call expects to receive a fax after establishing the connection.

`args.call_protocol_code`

Selects a calling protocol for the channel from one of the following (see *page 441* for descriptions):

```
CALL_PROTOCOL_FAX
CALL_PROTOCOL_FAX_NO_RAW
CALL_PROTOCOL_RAW
CALL_PROTOCOL_VOICE
CALL_PROTOCOL_VOICE_NO_RAW
```

The channel returns other detected call progress results after the timeout value set for the `ced_timeout` (the length of time to wait for the called station's ID signal) parameter.

The channel retrieves the `wait_for_ced` value from the *BT_CPARM.CFG* file or from the user-defined configuration file (see *Volume 6, Appendix A, Configuration Files*).

`args.func`

Pointer to a user-supplied integer function that the channel calls during call progress. The channel calls this user function in a loop until it returns a final call progress result, or the user function indicates termination of call progress by returning 1.

`args.timeout`

This value only applies if there is no other applicable timeout. It is used in the cases when *args.call_protocol* is either:

- `CALL_PROTOCOL_RAW`
- `CALL_PROTOCOL_VOICE` or `CALL_PROTOCOL_NO_RAW` and the user configuration parameter *v_timeout* is set to 0.

In such cases, it specifies the length of time to wait for a call to be established in milliseconds. 0 indicates to wait forever.

`args.sip_header_list_len`

Specifies the size of the memory buffer pointed to by the *sip_header_list* field. This field should be set when the Bfv application wants to retrieve the SIP header information specified in an inbound SIP 183 Session Progress response received during call setup.

This field should be set to the size of the memory buffer pointed to by the *sip_header_list* field. If the Bfv application does not want to retrieve the SIP header information specified in an inbound SIP 183 Session Progress response, this field should be set to 0 and *sip_header_list* should be set to NULL.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

`args.sip_header_list`

Specifies a pointer to a memory buffer allocated by the Bfv application to receive SIP header data from an inbound SIP 183 Session Progress response received during call setup. The size of the memory buffer pointed to by *sip_header_list* should be specified in the *sip_header_list_len* field.

This field should be set to the address of a memory buffer allocated by the Bfv application that will receive the SIP header data. As *sip_header_list* is a pointer to a *BT_SIP_HEADER_LIST* structure, the Bfv application memory buffer should be allocated and initialized in a manner similar to the following:

```
args.sip_header_list =
    (BT_SIP_HEADER_LIST *)malloc(1000);
memset(args.sip_header_list, 0, 1000);
args.sip_header_list_len = 1000;
```

If the Bfv application does not want to retrieve the SIP header information specified in an inbound SIP 183 Session Progress response, *sip_header_list_len* should be set to 0 and *sip_header_list* should be set to NULL.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

Note: The application must free this structure after use to release the memory.

Output

Return value:

`0` Timeout has expired

`<0` Error

`>0` Successful return

`args.cause`

Returns an ISDN-defined code if the call fails that provides the reason why the function failed to complete the outbound call. See *Volume 6, Appendix D, Defining ISDN Cause Codes* for a description.

`args.call_transport`

Returns a code indicating the transport protocol used for the outbound call. Valid values are:

TRANSPORT_TYPE_UDP

User Datagram Protocol (UDP) was the transport protocol used for the outbound SIP call.

TRANSPORT_TYPE_TCP

Transmission Control Protocol (TCP) was the transport protocol used for the outbound SIP call.

TRANSPORT_TYPE_DEFAULT

This code is returned for calls using the H.323 internet protocol and PSTN line types.

`args.subcause`

If the call fails, returns the response code of the SIP message that resulted in the failure of the outbound call. This field is only applicable to calls using the SIP protocol. The definitions of these response codes are specified in RFC 3261.

args.cause_location

Returns an ISDN-defined code that indicates the originator (local or remote) of the failure notification (*args.cause*). Valid values are:

IISDNlocUSER	User
IISDNlocPVT_LOCAL	Private network serving the local user
IISDNlocPUB_LOCAL	Public network serving the local user
IISDNlocTRANSIT_NET	Transit network
IISDNlocPUB_REMOTE	Public network serving the remote user
IISDNlocPVT_REMOTE	Private network serving the remote user
IISDNlocINTERNATIONAL	International network
IISDNlocBEY_INTERWORK	Network beyond internet-working point

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

args.cres.name_ident

Returns text in the *name_ident* field of the CALL_RES structure (see *Volume 6, Appendix B, CALL_RES Structure Parameters*), indicating the name of the calling party if provided by the network. The field allows a maximum of 50 characters (ECC_MAX_NAME_STR). This field is only valid for an E1 or T1 QSIG protocol.

args.cres.name_char_set

Indicates the international standard specification (ISOxxx) of the character set in use (only supported by a QSIG protocol). For values, see *Volume 6, Appendix B, CALL_RES Structure Parameters.*

args.cres.connected_num

Returns the telephone number of the connected party in the CALL_RES structure if the network provided this data (only supported by a QSIG protocol).

`args.sip_header_list_len`

If an outbound SIP call receives a SIP 183 Session Progress response, this field will be set to the amount of data in the *sip_header_list* memory buffer that has been populated with SIP header data. If the memory buffer pointed to by *sip_header_list* is not large enough to hold all the SIP header data from the inbound SIP 183 Session Progress response, then *sip_header_list_len* will be set to a value of *SIP_HEADER_INVALID_LEN*. In this case, the memory buffer pointed to by *sip_header_list* will only be populated with complete SIP header name/header value pairs that fit in the buffer.

For example, if there are 10 SIP headers in the inbound SIP 183 Session Progress response, but the memory buffer pointed to by *sip_header_list* can only hold enough data for 9 complete SIP header name/header value pairs retrieved from the SIP 183 Session Progress response with 30 bytes of the buffer unused, then *sip_header_list_len* will be set to a value of *SIP_HEADER_INVALID_LEN* and only 9 complete SIP header name/header value pairs will be returned in the *sip_header_list* buffer. No partial or incomplete SIP header information from the 10th header will be used to populate the remaining 30 bytes in the *sip_header_list* buffer.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

`args.sip_header_list`

If an outbound SIP call receives a 183 Session Progress response, the memory buffer pointed to by *sip_header_list* will be populated with a singly linked list of SIP header data stored in *BT_SIP_HEADER_NODE* structures starting with a *BT_SIP_HEADER_LIST* structure. The data structures used to access the SIP header data in the singly linked list are defined below:

```
struct BT_SIP_HEADER {
    char *header_name;
    char *header_value;
};
typedef struct _BT_SIP_HEADER_NODE {
    struct BT_SIP_HEADER header;
    struct _BT_SIP_HEADER_NODE *next_header;
} BT_SIP_HEADER_NODE;
```

```
typedef struct _BT_SIP_HEADER_LIST {
    int num_sip_headers;
    BT_SIP_HEADER_NODE sip_headers;
} BT_SIP_HEADER_LIST;
```

Some SIP headers that have multiple values may be returned as several SIP headers. For example, the following header:

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

Will be returned as five separate headers:

Header Name	*Header Value*
Allow	INVITE
Allow	ACK
Allow	OPTIONS
Allow	CANCEL
Allow	BYE

If some of the SIP header names in the inbound SIP 183 Session Progress response were specified in compact form, they may be returned to the Bfv application in long form (e.g., Content-Type header name specified in inbound SIP request as "c", returned to Bfv application as "Content-Type").

The maximum supported number of SIP headers that can be returned is 98.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

Details

BfvCallWaitForComplete waits for an answer to an outbound call initiated by using *BfvCallSetup*. You must use *BfvCallSetup* first to start the outbound call process.

Depending on the type of phone line and the value set for *args.call_protocol_mode*, *BfvCallWaitForComplete* might use in-band call progress analysis. However, if the protocol provides an out-of-band indication of call answer or call failure, this indication can cause *BfvCallWaitForComplete* to shorten or eliminate the use of in-band analysis.

BfvCallWaitForComplete can set the line state to either `OFF_HOOK` or `CONNECTED`. The application program must examine *args.res* to determine how to proceed. For example, if the channel

detects a busy indication, the application normally calls *BfvCallDisconnect* and *BfvCallWaitForRelease* to clear the channel.

For an example of processing the 183 Session Progress headers during call setup, see the *faxll.c* sample application.

See Also

BfvCallDisconnect, *BfvCallSetup*, *BfvCallWaitForAlerting*, *BfvCallWaitForRelease*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
args.phonenum = "13115552368";
BfvCallSetup(lp, &args);

BT_ZERO(args);
args.call_protocol_code = CALL_PROTOCOL_FAX;
args.timeout = 0;
BfvCallWaitForComplete(lp, &args);
if (args.res.status != BT_STATUS_OK)
{

    /* It did not connect */

}
```

BfvCallWaitForHold

Purpose

Waits for the Bfv API to finish transitioning to the hold state.

The SR140 and analog DID lines do not support this function.

Syntax

```
void
BfvCallWaitForHold          (lp, args)
   BTLINE                   *lp;
   struct args_cc           *args;
```

The structure contains the following fields.

Input Field

None

Output Field

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing an output field.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use the *BfvCallWaitForHold* function after calling *BfvCallHold* to allow the Bfv API to finish transitioning to the hold state.

See Also

BfvCallHold, *BfvCallRetrieve*, *BfvCallWaitForRetrieve*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
BfvCallHold(lp, &args);

BT_ZERO(args);
BfvCallWaitForHold(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* The API did not change to a hold state */
}
```

BfvCallWaitForRelease

Purpose

Finishes the process of terminating a telephone call.

Syntax

```
void
BfvCallWaitForRelease        (lp, args)
   BTLINE                    *lp;
   struct args_cc            *args;
```

The structure contains the following fields:

Input Field

```
int timeout;
```

Output Fields

```
int cause;
int cause_location;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.timeout

Specifies an integer value that determines the length of time to wait for the function to complete. Valid values are:

0	Indicates waiting forever with no timeout.
nonzero	Indicates the number of milliseconds to wait for the function to complete.

Output

Return value: None.

args.cause

Returns an ISDN-defined code if the call fails that provides the reason why the function failed to finish disconnecting the call. See *Volume 6, Appendix D, Defining ISDN Cause Codes* for a description.

`args.subcause`

Returns the response code of the SIP message that resulted in the termination of the call. This field is only applicable to calls using the SIP protocol. The definitions of these response codes are specified in RFC 3261.

`args.cause_location`

Returns an ISDN-defined code that indicates the originator (local or remote) of the failure notification (*args.cause*). For IP protocol calls (SIP or H.323), this field indicates which side initiated the termination of the call.

Valid values are:

`IISDNlocUSER`	User
`IISDNlocPVT_LOCAL`	Private network serving the local user
`IISDNlocPUB_LOCAL`	Public network serving the local user
`IISDNlocTRANSIT_NET`	Transit network
`IISDNlocPUB_REMOTE`	Public network serving the remote user
`IISDNlocPVT_REMOTE`	Private network serving the remote user
`IISDNlocINTERNATIONAL`	International network
`IISDNlocBEY_INTERWORK`	Network beyond internet-working point

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use *BfvCallWaitForRelease* after calling *BfvCallDisconnect* to allow the channel to finish disconnecting the call and clearing the line. An application can also use this function to wait for the other party to clear a call on protocols that provide remote disconnect notification.

You must use this function for all calls after calling *BfvCallDisconnect* to complete disconnection of the call. When the function terminates successfully, the line is ready to make another call.

If you use the high-level call control functions, use the *BfvLineTerminateCall* function to perform the equivalent terminating process.

Note: On protocols that support Advice Of Charge (AOC), successful completion of this function (or *BfvLineTerminateCall*) means that the application can obtain the AOC information provided by the network.

See Also

BfvCallDisconnect, *BfvLineTerminateCall*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
args.cause = IISDNcausDEFAULT;
BfvCallDisconnect(lp, &args);

BT_ZERO(args);
args.timeout = 0;
BfvCallWaitForRelease(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* It did not disconnect */
}
```

BfvCallWaitForRetrieve

Purpose

Waits for the Bfv API to finish transitioning out of the hold state.

The SR140 and analog DID lines do not support this function.

Syntax

```
void
BfvCallWaitForRetrieve      (lp, args)
   BTLINE                   *lp;
   struct args_cc           *args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing an output field.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

Use the *BfvCallWaitForRetrieve* function after calling *BfvCallRetrieve* to allow the Bfv API to finish transitioning out of the hold state.

See Also

BfvCallHold, *BfvCallRetrieve*, *BfvCallWaitForHold*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
BfvCallHold(lp, &args);

BT_ZERO(args);
BfvCallWaitForHold(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* The API did not change to the hold state */
}

BT_ZERO(args);
BfvCallRetrieve(lp, &args);

BT_ZERO(args);
BfvCallWaitForRetrieve(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* The API did not exit the hold state */
}
```

BfvCallWaitForSetup

Purpose

Waits to detect an incoming call enabled for ring detection.

Syntax

```
int
BfvCallWaitForSetup        (lp, args)
   BTLINE                  *lp;
   struct args_cc          *args;
```

The structure contains the following fields.

Input Field

```
int timeout;
unsigned sip_header_list_len;
BT_SIP_HEADER_LIST *sip_header_list;
```

Output Fields

```
unsigned char calling_party_presentation;
unsigned char calling_party_screening;
CALL_RES cres;
char orig_called_num[256];
RES res;
enum TRxTransportType call_transport;
unsigned sip_header_list_len;
BT_SIP_HEADER_LIST *sip_header_list;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.timeout

Specifies an integer value that determines the length of time to wait for the function to complete. Valid values are:

0	Indicates waiting forever with no timeout.
nonzero	Indicates the number of milliseconds to wait for the function to complete.

`args.sip_header_list_len`

Specifies the size of the memory buffer pointed to by the *sip_header_list* field. This field should be set when the Bfv application wants to retrieve the SIP header information specified in an inbound SIP INVITE request used to establish a SIP call.

This field should be set to the size of the memory buffer pointed to by the *sip_header_list* field.

If the Bfv application doesn't want to retrieve the SIP header information specified in an inbound SIP INVITE request, this field should be set to 0 and *sip_header_list* should be set to NULL.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

`args.sip_header_list`

Specifies a pointer to a memory buffer allocated by the Bfv application to receive SIP header data from an inbound SIP INVITE request used to establish a SIP call. The size of the memory buffer pointed to by *sip_header_list* should be specified in the *sip_header_list_len* field.

This field should be set to the address of a memory buffer allocated by the Bfv application that will receive the SIP header data. As *sip_header_list* is a pointer to a *BT_SIP_HEADER_LIST* structure, the Bfv application memory buffer should be allocated and initialized in a manner similar to the following:

```
args.sip_header_list =
      (BT_SIP_HEADER_LIST *)malloc(1000);
memset(args.sip_header_list, 0, 1000);
args.sip_header_list_len = 1000;
```

If the Bfv application doesn't want to retrieve the SIP header information specified in an inbound SIP INVITE request, *sip_header_list_len* should be set to 0 and *sip_header_list* should be set to NULL.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

Note: The application must free this structure after use to release the memory.

Output

Return value:

0	Timeout has expired
<0	Error
>0	Successful return

args.calling_party_presentation

Returns a value for ISDN calls that indicates the origin of the calling party number presented to the called party or the accessibility level of the calling party number (see *page 442* for definitions of values). Valid values are:

ECC_PRES_APP_DEFINED

ECC_PRES_ALLOWED

ECC_PRES_RESTRICTED

ECC_PRES_NUM_NOT_AVAIL

args.calling_party_screening

Returns a value for ISDN calls that indicates the origin and validity of the calling party number passed to the called party (see *page 442* for definitions of values). Valid values are:

ECC_SCRN_APP_DEFINED

ECC_SCRN_USER_NOT_SCREENED

ECC_SCRN_USER_VERIFICATION_PASSED

ECC_SCRN_USER_VERIFICATION_FAILED

ECC_SCRN_NETWORK_PROVIDED

args.cres.redir_number
args.cres.redir_reason

Returns a CALL_RES structure containing status information. See *CALL_RES Structure Parameters* in *Volume 6, Appendix B*.

When receiving a diverted incoming call on a port using the QSIG protocol, this field outputs the *redir_number* and *redir_reason* fields of the CALL_RES structure to indicate the phone number of the device diverting the call and the reason for diverting the call. Valid diversion reasons are:

DIVERT_NONE	Used for call that does not divert.
DIVERT_BUSY	Call diverted for busy condition.
DIVERT_UNCONDITIONAL	Call diverted without conditions.
DIVERT_NO_RESPONSE	Call diverted for unresponsive line.

args.cres.name_ident

Returns text in the *name_ident* field of the `CALL_RES` structure, indicating the name of the calling party if provided by the network. The field allows a maximum of 50 characters (`ECC_MAX_NAME_STR`). This field is only valid for an E1 or T1 QSIG protocol.

`args.cres.name_char_set`

Indicates the international standard specification (ISOxxx) of the character set in use (only supported by a QSIG protocol) in the *name_char_set* field of the `CALL_RES` structure. For values, see *Volume 6, Appendix B, CALL_RES Structure Parameters.*

`args.orig_called_num[256]`

Returns the number of the first destination for the outbound call. The field allows a maximum of 255 characters (`ECC_MAX_DIGIT_STR` - 1).

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

`args.call_transport`

Returns a code indicating the transport protocol used for the inbound call. Valid values are:

TRANSPORT_TYPE_UDP

User Datagram Protocol (UDP) was the transport protocol used for the inbound SIP call.

TRANSPORT_TYPE_TCP

Transmission Control Protocol (TCP) was the transport protocol used for the inbound SIP call.

TRANSPORT_TYPE_DEFAULT

This code is returned for calls using the H.323 internet protocol and PSTN line types.

`args.sip_header_list_len`

If a SIP call has successfully been received, this field will be set to the amount of data in the sip_header_list memory buffer that has been populated with SIP header data. If the memory buffer pointed to by *sip_header_list* isn't large enough to hold all the SIP header data from the inbound SIP INVITE request, then *sip_header_list_len* will be set to a value of *SIP_HEADER_INVALID_LEN*. In this case, the memory buffer pointed to by *sip_header_list* will only be populated with complete SIP header name/header value pairs that fit in the buffer.

For example, if there are 10 SIP headers in the inbound SIP INVITE request, but the memory buffer pointed to by *sip_header_list* can only hold enough data for 9 complete SIP header name/header value pairs retrieved from the SIP INVITE request with 30 bytes of the buffer unused, then *sip_header_list_len* will be set to a value of *SIP_HEADER_INVALID_LEN* and only 9 complete SIP header name/header value pairs will be returned in the *sip_header_list* buffer. No partial or incomplete SIP header information from the 10th header will be used to populate the remaining 30 bytes in the *sip_header_list* buffer.

Note: This field only works for inbound calls using the SIP internet protocol. The Bfv API ignores this field for all outbound calls and calls using the H.323 internet protocol or PSTN line types.

`args.sip_header_list`

If a SIP call has successfully been received, the memory buffer pointed to by *sip_header_list* will be populated with a singly linked list of SIP header data stored in *BT_SIP_HEADER_NODE* structures starting with a *BT_SIP_HEADER_LIST* structure. The data structures used to access the SIP header data in the singly linked list are defined below:

```
struct BT_SIP_HEADER {
    char *header_name;
    char *header_value;
};
typedef struct _BT_SIP_HEADER_NODE {
    struct BT_SIP_HEADER header;
    struct _BT_SIP_HEADER_NODE *next_header;
} BT_SIP_HEADER_NODE;
typedef struct _BT_SIP_HEADER_LIST {
    int num_sip_headers;
    BT_SIP_HEADER_NODE sip_headers;
} BT_SIP_HEADER_LIST;
```

Some SIP headers that have multiple values may be returned as several SIP headers. For example, the following header:

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

Will be returned as five separate headers:

Header Name	*Header Value*
Allow	INVITE
Allow	ACK
Allow	OPTIONS
Allow	CANCEL
Allow	BYE

If some of the SIP header names in the inbound SIP INVITE request were specified in compact form, they may be returned to the Bfv application in long form (e.g., Content-Type header name specified in inbound SIP request as "c", returned to Bfv application as "Content-Type").

The maximum supported number of SIP headers that can be returned is 98.

Note: This field only works for inbound calls using the SIP internet protocol. The Bfv API ignores this field for all outbound calls and calls using the H.323 internet protocol or PSTN line types.

Details

If the application previously enabled ring detection, *BfvCallWaitForSetup* waits for detection of an incoming call. Enable incoming ring detection by using the *BfvCallRingDetect* function with the value for *args.mode* set to TRUE.

When *BfvCallWaitForSetup* detects an incoming call, answer the call using *BfvCallAccept* followed by *BfvCallWaitForAccept*. To divert the incoming call, use the *BfvCallWaitForHold* functions, noting that a call cannot be diverted after it has been answered. This sequence of function calls ensures that the channel answers the call properly and completely before proceeding with any further call processing.

If you use the high-level call control functions, use the *BfvLineAnswer* function to perform the equivalent answering process.

See Also

BfvCallAccept, *BfvCallRingDetect*, *BfvCallWaitForAccept*, *BfvCallWaitForHold*, *BfvLineAnswer*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;

BT_ZERO(args);
args.mode = TRUE;
BfvCallRingDetect(lp, &args);

BT_ZERO(args);
args.timeout = 0;
BfvCallWaitForSetup(lp, &args);

BT_ZERO(args);
BfvCallAccept(lp, &args);

BT_ZERO(args);
args.timeout = 0;
BfvCallWaitForAccept(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* It did not answer */
}
```

BfvCallWaitTransferComplete

Purpose

Waits for the line to complete the call transfer command after the application initiates completion of the transfer.

The SR140 and analog DID lines do not support this function.

Syntax

```
void
BfvCallWaitTransferComplete(lp, args)
   BTLINE                  *lp;
   struct args_cc          *args;
```

The structure contains the following fields.

Input Fields

```
int disable_auto_sw_connect;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing an output field.

args.disable_auto_sw_connect

Specifies a Boolean value that indicates whether to turn the automatic switch connection feature on or off when completing a two-channel call transfer that requires connecting the two B channels together (QSIG protocol) so that the parties can communicate while executing the transfer. Set this field as follows:

FALSE	Automatically makes the switch connection for a call transfer that requires two B channels connected together.
TRUE	Turns off the automatic switch connection capability for the two-channel call transfer. The application must call *BfvCallSwitchConnect* to connect the B channels together.

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use *BfvCallTransferComplete* to start completing a call transfer but this function does not wait for the transfer command to complete. To complete the transfer, use this *BfvCallWaitTransferComplete* function to wait for the transfer process to finish and then call the *BfvCallWaitForRelease* function to complete the full transfer procedure.

When using a QSIG protocol that requires connecting two B channels together to make a call transfer, this function automatically connects the B channels when you set the *args.disable_auto_sw_connect* to `FALSE`. If you set *args.disable_auto_sw_connect* to `TRUE`, your application must connect the two B channels together to make a call transfer by using the *BfvCallSwConnect* function (see *Volume 1, Administration, Management, and Configuration*).

See Also

BfvCallTransferComplete, *BfvCallWaitForRelease*

Example

```
BTLINE *lp;
.
.
.
struct args_cc args;
.
. /* Transfer Set up */
.
BT_ZERO(args);
BfvCallTransferComplete(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    BT_ZERO(args);
    BfvCallDisconnect(lp,&args);
}
else
    BT_ZERO(args);
    BfvCallWaitTransferComplete (lp,&args);
}

BT_ZERO(args);
BfvCallWaitForRelease(lp, &args);
```

BfvLineAnswer

Purpose

Answers an incoming call, sets the line state to CONNECTED and waits for answer confirmation before returning.

Syntax

```
void
BfvLineAnswer                (lp, args)
   BTLINE                    *lp;
   struct args_telephone     *args;
```

The structure contains the following fields.

Input Fields

```
long timeout;
int orig_answer;
```

Output Fields

```
int cause_code;
int subcause;
int cause_location;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.timeout

Specifies an integer value that determines the length of time to wait for the function to complete. Valid values are:

0	Indicates waiting forever with no timeout.
nonzero	Indicates the number of milliseconds to wait for the function to complete.

`args.orig_answer`

Specifies whether off-hook is for the purpose of originating a call (`BT_ORIGINATE`) or answering a call (`BT_ANSWER`). Optionally, this field can be left set to `0`. It is extremely rare for an application to ever set this input to any value other than `0` or `BT_ORIGINATE`. In ISDN mode, this argument is ignored.

Output

Return value: None.

`args.cause_code`

Returns an ISDN-defined code if the call fails that provides the reason why the function failed to answer and connect the call. See *Volume 6, Appendix D, Defining ISDN Cause Codes* for a description.

`args.subcause`

Returns the response code of the SIP message that resulted in the failure of the inbound call. This field is only applicable to calls using the SIP protocol. The definitions of these response codes are specified in RFC 3261.

`args.cause_location`

Returns an ISDN-defined code that indicates the originator (local or remote) of the failure notification (args.cause_code). Valid values are:

IISDNlocUSER	User
IISDNlocPVT_LOCAL	Private network serving the local user
IISDNlocPUB_LOCAL	Public network serving the local user
IISDNlocTRANSIT_NET	Transit network
IISDNlocPUB_REMOTE	Public network serving the remote user
IISDNlocPVT_REMOTE	Private network serving the remote user
IISDNlocINTERNATIONAL	International network
IISDNlocBEY_INTERWORK	Network beyond internet-working point

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Typically, an application calls *BfvLineAnswer* after calling the *BfvLineWaitForCall* function to establish the reason for connecting the line. This function can also be used by itself when you connect the channel to the H.100 bus or when you do not connect the channel to a phone line.

If you use the low-level call control functions, use the *BfvCallAccept* and *BfvCallWaitForAccept* functions to perform the equivalent answering process.

See Also

BfvCallAccept, *BfvCallWaitForAccept*, *BfvLineTerminateCall*, *BfvLineWaitForCall*

Example

```
BTLINE *lp;
struct args_telephone args;

BT_ZERO(args);
BfvLineAnswer (lp, &args);
```

BfvLineCCProtocolGet

Purpose

Retrieves the protocol assigned to the module.

Syntax

```
TRxPortConfig
BfvLineCCProtocolGet        (lp, args)
   BTLINE                   *lp;
   struct args_telephone    *tel_args;
```

The structure contains the following fields.

Input Field

None

Output Fields

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing output fields.

Output

Returns one of the following values from *TRxPortConfig*:

```
0  PORT_CONFIG_INACTIVE (Port disabled)
2  PORT_CONFIG_H1X0
3  PORT_CONFIG_T1_ISDN
4  PORT_CONFIG_T1_ROBBED_BIT
5  PORT_CONFIG_E1_ISDN
6  PORT_CONFIG_E1_CAS
7  PORT_CONFIG_E1_CAS_R2
8  PORT_CONFIG_E1_DPNSS
9  PORT_CONFIG_BRI
10 PORT_CONFIG_ANALOG
11 PORT_CONFIG_ANALOG_DID
12 PORT_CONFIG_HOST_MODULE_1
13 PORT_CONFIG_HOST_MODULE_2
14 PORT_CONFIG_HOST_MODULE_3
15 PORT_CONFIG_HOST_MODULE_4
16 PORT_CONFIG_HOST_MODULE_5
17 PORT_CONFIG_HOST_MODULE_6
18 PORT_CONFIG_HOST_MODULE_7
19 PORT_CONFIG_HOST_MODULE_8
20 PORT_CONFIG_HOST_MODULE_9
21 PORT_CONFIG_E1_QSIG
22 PORT_CONFIG_T1_QSIG
23 PORT_CONFIG_BRI_QSIG
24 PORT_CONFIG_UNKNOWN
```

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

An application uses this function to determine the protocol configured for a given channel. With this information, an application can then provide appropriate Bfv API call control functions with the ability to:

- Process protocol-specific dial strings, or
- Establish protocol-specific passthrough parameters for the Bfv API to simply pass to and from an integrated IP call control stack without modification.

Example

```
BTLINE *lp;
struct args_telephone args;

TRxPortConfig config;

BT_ZERO(args);
config = BfvLineCCProtocolGet(lp, &args);
if (args.res.status != BT_STATUS_OK)
{
    /* Failed to retrieve the protocol */
}
```

BfvLineDialString

Purpose

Places the line in an OFF_HOOK state, dials the digits specified, and returns after dialing the last digit.

Syntax

```
int
BfvLineDialString          (lp, args)
   BTLINE                  *lp;
   struct args_telephone   *args;
```

The structure contains the following fields.

Input Field

```
char *phonenum;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.phonenum

Specifies a pointer to a null-terminated ASCII string that identifies the phone number to dial. Dial strings (phone numbers) may be up to a maximum of 255 characters (ECC_MAX_DIGIT_STR - 1).

PSTN Telephony

The dial string field supports the following digits and control characters. Invalid characters are ignored; upper and lower case letters are equivalent. Some protocols ignore control characters and only accept DTMF characters.

Valid Digits and Control Characters

0 - 9	Dials digits ‘0’ through ‘9’.
# (pound)	Dials a pound.
* (asterisk)	Dials an asterisk.
A - D	Sends the DTMF tone corresponding to the specified alphabetic character.
p	Changes the current or default dialing mode from tone dialing to pulse dialing.
t	Changes the current or default dialing mode from pulse dialing to tone dialing.
w	Waits for dial tone.
, (comma)	Causes a 1-second pause.
; (semicolon), i or I	Causes a 5-second pause. To create longer pauses, string any of these characters together.
! (exclamation point)	Sends a hook flash on analog and T1 robbed bit modules.

Note: In an analog environment or when using a T1 robbed bit FXS loopstart or E1 CAS loopstart protocol, the ‘w’ character means wait for dial tone. All other protocols ignore the ‘w’ and ‘i’ characters. Only analog environments and T1 robbed bit or E1 CAS protocols use the ‘p’, ‘t’, comma and semicolon characters.

IP Telephony

For IP outbound calls using the H.323 protocol:

- `Phone#@IP Address:Port#`

  If the receiving side does not require a phone number, a value for `Phone#` is optional. Also, `:Port#` is optional — the Bfv API uses 1720 as the default port value.

- `TA:IP Address:Port#,Phone#`

  `:Port#` is optional — the Bfv API uses 1720 as the default port value. If the receiving side does not require a phone number, a value for `Phone#` is optional.

- `Name:<name of person to dial>`

  Use this form only if using gatekeeper

- `E164alias:7894561234`

  Use this form only if using gatekeeper

Note: DNS lookups are not supported in H.323. You can use an H.323 or E.164 alias in conjunction with a gatekeeper to provide similar functionality.

Examples

```
4082345555@10.155.89.6:175
4082345555@10.155.89.6

TA:10.155.89.6:175,4082345555
TA:10.155.89.6,4082345555
TA:10.155.89.6

Name:Fred Smith
E164alias:4082345555
```

For IP outbound calls using the SIP protocol:

```
Phone#@IP Address:Port#
```

If the receiving side does not require a phone number, a value for Phone# is optional. Also, :Port# is optional - the Bfv API uses 5060 as the default port value.

Examples (IPv4 Addresses)

```
4082345555@10.155.89.7:175
4082345555@10.155.89.7
```

Examples (IPv6 Addresses)

```
4082345555@[2000::2ef3:1dff:ea3]:175
4082345555@[fe80::1f4:189c:74da:69f7]
```

Note: IPv6 addresses must be enclosed in brackets. In addition, if a link-local IPv6 address is specified, the Scope ID should be omitted from the address.

DTMF Post Dialing

For all types of IP calls, the character '&' (ampersand) may be included to initiate post-dialing. This character indicates that the rest of phonenum specifies a sequence of DTMF digits to be "post-dialed" after messages from the remote side indicate the call is proceeding towards connecting.

Within the post-dial string, all dialing characters listed for PSTN Telephony are allowed except for 'p', 't', 'w', and '!'. The appearance of an additional '&' will terminate processing of the string.

Post-dialing of the specified digits will occur upon the first receipt of one of the following IP call control messages:

- SIP -- 183 Progress
- SIP -- 200 OK
- H.323 -- Progress
- H.323 -- Connect

The post-dial feature is controlled by the user configuration file parameter *post_dialing_enable*. If the feature is disabled by that parameter, then the ampersand has no special effect and the entire phonenum field is used as is.

Note: If a program accepts a phone number on the command line, the phone number will likely need to be quoted if it contains an ampersand, since this is a special character on most OSes. (Use double quotes (") on Windows, or single (') or double (") quotes on Linux.)

Output

Return value:

`<0`	Ringing detected or other error; dialing aborted.
`0`	Dialing completed; no errors.
`1`	No dial tone detected.
`2`	No loop current detected.
`3`	Local phone in use (country specific).
`4`	Trunk is busy (e.g., out dialing on a PBX system).
`7`	Dial tone detected after completing dialing.
`8`	T1 time slot busy.
`9`	Call collision detected (ringing occurred while dialing).
`10`	No wink signal on second or later 'w'.
`11`	ISDN invalid dial string.
`12`	Failure attempting a redial too soon because of Japanese restrictions.

Note: To determine the corresponding `res.line_status` values for `BT_STATUS_ERROR_DIAL`, returned by the *BfvLineOriginateCall* function (see *page 385*), add each of the above return values to 257 (`DIAL_OK`). For more details, see *Result Structures* in *Volume 6, Appendix B*.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function only serves to provide backward compatibility for existing applications that call it to place an outgoing call. You should verify that new applications place outgoing calls using either the *BfvLineOriginateCall* (see *page 374*) function or the low-level combination of the *BfvCallSetup* (see *page 303*) and *BfvCallWaitForComplete* (see *page 337*) functions.

For applications using the *BfvLineDialString* function, the function sets the line state to OFF_HOOK after successfully dialing.

For applications using Euro-ISDN on E1 or BRI lines, this function automatically sends a dial string using the protocol's overlapped dialing feature when the phone number exceeds 20 digits. The Euro-ISDN protocol only allows applications to send 20 digits in a block when placing a call. For phone numbers exceeding 20 digits, the protocol uses a process called overlapped dialing. This process sends extra digits after the initial call setup, allowing:

- The application to dial very large phone numbers
- The remote end to start answering a call before it receives all the digits

Users can place a call with up to 255 digits in the dial string. The Bfv API automatically breaks up the dial string into multiple blocks of 20 digits, and uses the overlapped dialing feature in the protocol to send one block of digits at a time. This process does not require any changes in the application.

The function of *BfvLineDialString* does not perform any call progress on the line. To perform call progress after the successful return of *BfvLineDialString*, refer to the following in *Volume 3*:

- *BfvLineCallProgressEnable*
- *BfvLineCallProgressDisable*
- *BfvDataCP*

To set the line state to a connected state after satisfactory call progress, use the LINE_STATE macro. See *Volume 1, Chapter 1, The BTLINE Structure.*

International Issues

In some countries, the PTT imposes certain dialing restrictions. Applications dialing a fax machine in any of these countries must use either the *BfvLineOrigCallDB* function or the *BfvDialDBCheck* function in conjunction with the *BfvDialDBUpdate* function. Otherwise, the application might be non-compliant with the target country's regulations.

For more information, see either:

- *Dialing Database Functions on page 420*
- *Country-Specific Parameter Files on page 1439*

See Also

BfvCallSetup, *BfvCallWaitForComplete*, *BfvLineOriginateCall*

Example

```
BTLINE *lp;
int dial_result;
struct args_telephone args;

BT_ZERO(args);
args.phonenum = "17814494100";
dial_result = BfvLineDialString(lp, &args);
if (dial_result != 0)
    printf("Error in Dialing\n");
```

BfvLineOriginateCall

Purpose

Places a phone call on an outgoing line.

Analog DID lines do not support this function.

Syntax

```
void
BfvLineOriginateCall        (lp, args)
   BTLINE                   *lp;
   struct args_telephone    *args;
```

The structure contains the following fields.

Input Fields

```
char *arg;
int call_mode;
int call_protocol_code;
int (*func)(BTLINE *lp, char *arg);
char *phonenum;
long timeout;
enum TRxTransportType call_transport;
unsigned num_user_sip_headers;
struct BT_USER_SIP_HEADER *user_sip_headers;
int fax_media_feature_tag;
unsigned fallback_rtp_reinvite;
unsigned sip_header_list_len;
BT_SIP_HEADER_LIST *sip_header_list;
```

Output Fields

```
RES res;
int cause_code;
int subcause;
int cause_location;
enum TRxTransportType call_transport;
unsigned sip_header_list_len;
BT_SIP_HEADER_LIST *sip_header_list;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

`args.arg`

Provides an input argument for the *func* feature field. This field accepts a `NULL` setting if the user-defined function does not need an argument.

When processing SIP 183 Session Progress response headers, the callback function must have access to the *sip_header_list* information. This is done by setting *args.arg* as follows:

```
args.arg = (char *)&args;
```

`args.call_mode`

Specifies a Boolean value that indicates whether the outbound call expects to receive a fax transmission when the connection completes. Set the field as follows:

`FALSE`

Indicates that the outbound call does not expect to receive a fax.

`TRUE`

Indicates that the outbound call expects to receive a fax after establishing the connection.

`args.call_protocol_code`

Selects a calling protocol for the channel from one of the following:

`CALL_PROTOCOL_FAX`

Selects the fax protocol. Requests the channel to report the results from the channel's call progress analysis along with the raw data. The channel reports the results as soon as it establishes the fax connection or encounters a busy condition.

Other detected call progress results are returned after the `ced_timeout` (the length of time to wait for the called stations's id signal) time-out.

`CALL_PROTOCOL_VOICE`

Selects the voice protocol. Requests the module to report the results from the channel's call progress analysis along with the raw data. The channel reports the results as soon as it detects a human or other answer condition.

`CALL_PROTOCOL_RAW`

Requests the channel to report the raw HIGH/LOW call progress results without performing any analysis.

`CALL_PROTOCOL_FAX_NO_RAW`
`CALL_PROTOCOL_VOICE_NO_RAW`

Selects either the fax or voice protocol, requesting the channel to report the results from the channel's call progress analysis without including the raw data.

`args.func`

Specifies a pointer to a user-supplied integer function that the channel calls during call progress. The channel calls this user function in a loop until it returns a final call progress result, or the user function indicates termination of call progress by returning 1.

The *args.func* field is called as `(*args.func)(lp,args.arg)`. The *lp* argument contains the pointer to the line structure; the *args.arg* argument contains the supplied user-defined argument.

`args.phonenum`

Specifies a pointer to a null-terminated ASCII string that identifies the phone number to dial. Dial strings (phone numbers) may be up to a maximum of 255 characters (ECC_MAX_DIGIT_STR - 1) for most protocols and locations. See Details.

PSTN Telephony

The dial string field supports the following digits and control characters. Invalid characters are ignored; upper and lower case letters are equivalent. Some protocols ignore control characters and only accept DTMF characters.

Valid Digits and Control Characters

0 - 9	Dials digits ‘0’ through ‘9’.
# (pound)	Dials a pound.
* (asterisk)	Dials an asterisk.
A - D	Sends the DTMF tone corresponding to the specified alphabetic character.
p	Changes the current or default dialing mode from tone dialing to pulse dialing.
t	Changes the current or default dialing mode from pulse dialing to tone dialing.
w	Waits for dial tone.
, (comma)	Causes a 1-second pause.
; (semicolon), i or I	Causes a 5-second pause. To create longer pauses, string any of these characters together.
! (exclamation point)	Sends a hook flash on analog and T1 robbed bit modules.

Note: In an analog environment or when using a T1 robbed bit FXS loopstart or E1 CAS loopstart protocol, the ‘w’ character means wait for dial tone. All other protocols ignore the ‘w’ and ‘i’ characters. Only analog environments and T1 robbed bit or E1 CAS protocols use the ‘p’, ‘t’, comma and semicolon characters.

IP Telephony

For IP outbound calls using the H.323 protocol:

- `Phone#@IP Address:Port#`

  If the receiving side does not require a phone number, a value for `Phone#` is optional. Also, `:Port#` is optional — the Bfv API uses 1720 as the default port value.

- `TA:IP Address:Port#,Phone#`

  `:Port#` is optional — the Bfv API uses 1720 as the default port value. If the receiving side does not require a phone number, a value for `Phone#` is optional.

- `Name:<name of person to dial>`

  Use this form only if using gatekeeper

- `E164alias:7894561234`

Use this form only if using gatekeeper

Note: DNS lookups are not supported in H.323. You can use an H.323 or E.164 alias in conjunction with a gatekeeper to provide similar functionality.

Examples

```
4082345555@10.155.89.6:175
4082345555@10.155.89.6

TA:10.155.89.6:175,4082345555
TA:10.155.89.6,4082345555
TA:10.155.89.6

Name:Fred Smith
E164alias:4082345555
```

For IP outbound calls using the SIP protocol:

```
Phone#@IP Address:Port#
```

If the receiving side does not require a phone number, a value for Phone# is optional. Also, :Port# is optional - the Bfv API uses 5060 as the default port value.

Examples (IPv4 Addresses)

```
4082345555@10.155.89.7:175
4082345555@10.155.89.7
```

Examples (IPv6 Addresses)

```
4082345555@[2000::2ef3:1dff:ea3]:175
4082345555@[fe80::1f4:189c:74da:69f7]
```

Note: IPv6 addresses must be enclosed in brackets. In addition, if a link-local IPv6 address is specified, the Scope ID should be omitted from the address.

DTMF Post Dialing

For all types of IP calls, the character '&' (ampersand) may be included to initiate post-dialing. This character indicates that the rest of phonenum specifies a sequence of DTMF digits to be "post-dialed" after messages from the remote side indicate the call is proceeding towards connecting.

Within the post-dial string, all dialing characters listed for PSTN Telephony are allowed except for 'p', 't', 'w', and '!'. The appearance of an additional '&' will terminate processing of the string.

Post-dialing of the specified digits will occur upon the first receipt of one of the following IP call control messages:

- SIP -- 183 Progress
- SIP -- 200 OK
- H.323 -- Progress
- H.323 -- Connect

The post-dial feature is controlled by the user configuration file parameter *post_dialing_enable*. If the feature is disabled by that parameter, then the ampersand has no special effect and the entire phonenum field is used as is.

Note: If a program accepts a phone number on the command line, the phone number will likely need to be quoted if it contains an ampersand, since this is a special character on most OSes. (Use double quotes (") on Windows, or single (') or double (") quotes on Linux.)

`args.timeout`

This value only applies if there is no other applicable timeout. It is used in the cases when *args.call_protocol* is either:

- `CALL_PROTOCOL_RAW`
- `CALL_PROTOCOL_VOICE` or `CALL_PROTOCOL_NO_RAW` and the user configuration parameter *v_timeout* is set to 0.

In such cases, it specifies the length of time to wait for a call to be established in milliseconds. 0 indicates to wait forever.

`args.call_transport`

Selects the transport protocol to use for an outbound SIP call from one of the following:

TRANSPORT_TYPE_UDP

Select User Datagram Protocol (UDP) as the transport protocol to use for the outbound SIP call.

TRANSPORT_TYPE_TCP

Select Transmission Control Protocol (TCP) as the transport protocol to use for the outbound SIP call. In order to use this setting, TCP protocol support must be enabled in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

TRANSPORT_TYPE_DEFAULT

Use the default call transport, either UDP or TCP for the outbound SIP call. If the default call transport is not explicitly specified in the call control configuration file, UDP will be used.

Otherwise, the transport protocol used for the outbound SIP call will be what's specified in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

NOTE: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

`args.num_user_sip_headers`

Specifies the number of entries in the array of *BT_USER_SIP_HEADER* structures referenced by the *args.user_sip_headers* field. The maximum number of entries in the *BT_USER_SIP_HEADER* structure array is *BT_USER_SIP_HEADERS_MAX_NUM_HEADERS*.

NOTE: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

`args.user_sip_headers`

Specifies a reference to an array of *BT_USER_SIP_HEADER* structures that specify SIP header names and values to add to the initial SIP INVITE of an outbound call. The *BT_USER_SIP_HEADER* structure is defined below:

```
struct BT_USER_SIP_HEADER {
    char *header_name;
    char *header_value;
};
```

The number of entries in the array of *BT_USER_SIP_HEADER* structures referenced by the *args.user_sip_headers* field is specified by the value of the *num_user_sip_headers* field and can have a maximum size of *BT_USER_SIP_HEADERS_MAX_NUM_HEADERS*. If the value specified by *num_user_sip_headers* is 0, this field is ignored.

The *header_name* field in the *BT_USER_SIP_HEADER* structure points to a null-terminated ASCII string that identifies the name of the SIP header to add to the initial SIP INVITE for an outbound SIP call. The maximum length that can be specified for the header name (including the null terminator) is *BT_USER_SIP_HEADERS_MAX_NAME_LEN*.

The *header_value* field in the *BT_USER_SIP_HEADER* structure points to a null-terminated ASCII string that identifies the value of the SIP header to add to the initial SIP INVITE for

an outbound SIP call. The maximum length that can be specified for the header value (including the null terminator) is *BT_USER_SIP_HEADERS_MAX_VALUE_LEN*.

Applications should not attempt to specify standard SIP headers (e.g. To, From, Via, Call-ID, CSeq, Contact, etc.) using the *args.user_sip_headers* field as this may result in unpredictable and unsupported behavior.

NOTE: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

`args.fax_media_feature_tag`

Specifies a "sip.fax" media feature tag value to add to an Accept-Contact header in the outbound SIP INVITE request. Set this field to one of the following values:

BT_FAX_MEDIA_FEATURE_TAG_DEFAULT

Set "sip.fax" media feature tag to a default value based on the *fax_transport_protocol* parameter value in the *t38parameters* section of the *callctrl.cfg* file as specified in the following table:

fax_transport_protocol value	"sip.fax" value
t38_never	passthrough
t38_only	t38
t38_first	t38
Not specified in *callctrl.cfg* file	t38

BT_FAX_MEDIA_FEATURE_TAG_T38

Set" sip.fax" media feature tag to "t38".

BT_FAX_MEDIA_FEATURE_TAG_PASSTHROUGH

Set "sip.fax" media feature tag to "passthrough".

BT_FAX_MEDIA_FEATURE_TAG_DISABLED

Do not add "sip.fax" media feature tag to transmitted SIP INVITE message.

If a *fax_media_feature_tag* value is specified that is not supported by the current configuration (e.g., *BT_FAX_MEDIA_FEATURE_TAG_T38* specified but *fax_transport_protocol* value set to *t38_never* in the *callctrl.cfg* configuration file), the outbound SIP call will fail with the *status*

field in the RES results structure set to *BT_STATUS_ERROR* and the *line_status* field in the RES results structure set to *APIERR_CALL_CONTROL*.

In order to use this field for outbound SIP calls, RFC 6913 feature support must be enabled in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

`args.fallback_rtp_reinvite`

Specifies whether or not a SIP RTP reINVITE should be transmitted for G.711 fallback mode if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable). Valid values are:

BT_FALLBACK_RTP_REINVITE_DEFAULT

Default to the setting specified in the call control configuration file.

BT_FALLBACK_RTP_REINVITE_DISABLE

Do not transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

BT_FALLBACK_RTP_REINVITE_ENABLE

Transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

When the application sets the value in this field to *BT_FALLBACK_RTP_REINVITE_DEFAULT*, the functionality will default to the setting specified in the call control configuration file (*callctrl.cfg*) by the *g711_fallback_rtp_reinvite* parameter. If the *g711_fallback_rtp_reinvite* parameter isn't specified in the call control configuration file, then the default functionality is *BT_FALLBACK_RTP_REINVITE_DISABLE*.

An application can override the value specified in the call control configuration file by the *g711_fallback_rtp_reinvite* parameter by setting this field to a value of either *BT_FALLBACK_RTP_REINVITE_DISABLE* or *BT_FALLBACK_RTP_REINVITE_ENABLE*.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_DISABLE* will prevent transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_ENABLE* will result in transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable) and the fax transport protocol (*fax_transport_protocol*) parameter specified in the call control configuration file is set to *t38_first*. The SDP settings in the SIP RTP reINVITE will be the same RTP codec settings initially used to establish the call.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol or PSTN line types.

`args.sip_header_list_len`

Specifies the size of the memory buffer pointed to by the *sip_header_list* field. This field should be set when the Bfv application wants to retrieve the SIP header information specified in an inbound SIP 183 Session Progress response received during call setup.

This field should be set to the size of the memory buffer pointed to by the *sip_header_list* field. If the Bfv application does not want to retrieve the SIP header information specified in an inbound SIP 183 Session Progress response, this field should be set to 0 and *sip_header_list* should be set to NULL.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

`args.sip_header_list`

Specifies a pointer to a memory buffer allocated by the Bfv application to receive SIP header data from an inbound SIP 183 Session Progress response received during call setup. The size of the memory buffer pointed to by *sip_header_list* should be specified in the *sip_header_list_len* field.

This field should be set to the address of a memory buffer allocated by the Bfv application that will receive the SIP header data. As *sip_header_list* is a pointer to a *BT_SIP_HEADER_LIST* structure, the Bfv application memory buffer should be allocated and initialized in a manner similar to the following:

```
args.sip_header_list =
    (BT_SIP_HEADER_LIST *)malloc(1000);
memset(args.sip_header_list, 0, 1000);
args.sip_header_list_len = 1000;
```

If the Bfv application does not want to retrieve the SIP header information specified in an inbound SIP 183 Session Progress response, *sip_header_list_len* should be set to 0 and *sip_header_list* should be set to NULL.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

Note: The application must free this structure after use to release the memory.

Output

Return value: None.

`args.cause_code`

Returns an ISDN-defined code if the call fails that provides the reason why the function failed to complete the outbound call. See *Volume 6, Appendix D, Defining ISDN Cause Codes* for a description.

`args.subcause`

If the call fails, returns the response code of the SIP message that resulted in the failure of the outbound call. This field is only applicable to calls using the SIP protocol. The definitions of these response codes are specified in RFC 3261.

`args.cause_location`

IISDNlocUSER	User
IISDNlocPVT_LOCAL	Private network serving the local user
IISDNlocPUB_LOCAL	Public network serving the local user
IISDNlocTRANSIT_NET	Transit network
IISDNlocPUB_REMOTE	Public network serving the remote user
IISDNlocPVT_REMOTE	Private network serving the remote user
IISDNlocINTERNATIONAL	International network
IISDNlocBEY_INTERWORK	Network beyond internet-working point

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Table 15. Status Information Returned for args.res

args.res.status	**args.res.line_status**[1]
BT_STATUS_OK	DIAL_OK FCP_ANSWER FCP_ANSWER_TONE_DETECT FCP_BUSY1 FCP_BUSY2 FCP_CONFIRM FCP_DIALTON FCP_G2DETCT FCP_HUMAN FCP_PULSE FCP_QUIET FCP_RECALL FCP_RNGNOANS FCP_ROBUSY FCP_SILENCE FCP_UNKNOWN FCP_CNGDETCT FCP_ISDN_CALL_COLLISION FCP_ISDN_CALL_PROGRESS FCP_SITINTC FCP_SITNOCIR FCP_SITREORD FCP_SITVACODE
BT_STATUS_ERROR_DIAL	DIAL_CALL_COLLISION DIAL_ISDN_INVAL DIAL_JP_REDIAL_FAIL DIAL_LOCAL_IN_USE DIAL_NO_DIAL_TONE DIAL_NO_LOOP_CUR DIAL_NO_WINK DIAL_SLOT_BUSY DIAL_TRUNK_BUSY DIAL_TRANSORT_INVAL DIAL_UNKNOWN

1 See *Call Progress Notes on page 1418* for code descriptions.

`args.call_transport`

Returns a code indicating the transport protocol used for the outbound call. Valid values are :

TRANSPORT_TYPE_UDP

User Datagram Protocol (UDP) was the transport protocol used for the outbound SIP call.

TRANSPORT_TYPE_TCP

Transmission Control Protocol (TCP) was the transport protocol used for the outbound SIP call.

TRANSPORT_TYPE_DEFAULT

This code is returned for calls using the H.323 internet protocol and PSTN line types.

`args.sip_header_list_len`

If an outbound SIP call receives a SIP 183 Session Progress response, this field will be set to the amount of data in the *sip_header_list* memory buffer that has been populated with SIP header data. If the memory buffer pointed to by *sip_header_list* is not large enough to hold all the SIP header data from the inbound SIP 183 Session Progress response, then *sip_header_list_len* will be set to a value of *SIP_HEADER_INVALID_LEN*. In this case, the memory buffer pointed to by *sip_header_list* will only be populated with complete SIP header name/header value pairs that fit in the buffer.

For example, if there are 10 SIP headers in the inbound SIP 183 Session Progress response, but the memory buffer pointed to by *sip_header_list* can only hold enough data for 9 complete SIP header name/header value pairs retrieved from the SIP 183 Session Progress response with 30 bytes of the buffer unused, then *sip_header_list_len* will be set to a value of *SIP_HEADER_INVALID_LEN* and only 9 complete SIP header name/header value pairs will be returned in the *sip_header_list* buffer. No partial or incomplete SIP header information from the 10th header will be used to populate the remaining 30 bytes in the *sip_header_list* buffer.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

args.sip_header_list

If an outbound SIP call receives a 183 Session Progress response, the memory buffer pointed to by *sip_header_list* will be populated with a singly linked list of SIP header data stored in *BT_SIP_HEADER_NODE* structures starting with a *BT_SIP_HEADER_LIST* structure. The data structures used to access the SIP header data in the singly linked list are defined below:

```
struct BT_SIP_HEADER {
    char *header_name;
    char *header_value;
};
typedef struct _BT_SIP_HEADER_NODE {
    struct BT_SIP_HEADER header;
    struct _BT_SIP_HEADER_NODE *next_header;
} BT_SIP_HEADER_NODE;
typedef struct _BT_SIP_HEADER_LIST {
    int num_sip_headers;
    BT_SIP_HEADER_NODE sip_headers;
} BT_SIP_HEADER_LIST;
```

Some SIP headers that have multiple values may be returned as several SIP headers. For example, the following header:

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

Will be returned as five separate headers:

Header Name	*Header Value*
Allow	INVITE
Allow	ACK
Allow	OPTIONS
Allow	CANCEL
Allow	BYE

If some of the SIP header names in the inbound SIP 183 Session Progress response were specified in compact form, they may be returned to the Bfv application in long form (e.g., Content-Type header name specified in inbound SIP request as "c", returned to Bfv application as "Content-Type").

The maximum supported number of SIP headers that can be returned is 98.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

Details

This function performs the following:

- Places a phone call on an outgoing line.
- Dials a phone number and reports call progress results.
- Accepts a call protocol code that differentiates between a call answered by a fax machine and a call answered by a person and alters the behavior of the line accordingly.

The firmware performs sophisticated call progress functions.

In `CALL_PROTOCOL_RAW` mode, the channel reports the output of the call progress filter (a 300 – 600 Hz bandpass filter on each channel) and the time the output was in that particular state only. In this mode, the channel disables call analysis and provides raw call progress signal data directly to the user. The user then has complete control of signal interpretation.

In `CALL_PROTOCOL_FAX` or `CALL_PROTOCOL_VOICE` mode, the channel continues to report the `CALL_PROTOCOL_RAW` output values, which can be analyzed by the user-supplied function, and it also analyzes those values for meaningful patterns, such as ring-back, busy, and answer. If a final call progress value is detected (see *args.res*), call progress is halted.

Because connection to a fax machine is the goal in fax mode, some results are suppressed until the end of the `ced_timeout`.

For example, if a person answers a phone call, realizes the beeping on the line is a fax machine, and switches on the fax machine, fax mode suppresses the `FCP_ANSWER` or `FCP_HUMAN` result and reports `FCP_ANSWER_TONE_DETECT` when the fax machine is switched on. If the remote fax machine is not detected and the `ced_timeout` expires, `FCP_RNGNOANS` or `FCP_ANSWER` (or a similar condition) is reported, depending on the conditions detected.

BfvLineOriginateCall sets the line state to either `OFF_HOOK` or `CONNECTED`. The application program must examine *args.res* to determine how to proceed. For example, if the channel detects a busy indication, the application normally calls *BfvLineTerminateCall* to end the call.

BfvLineOriginateCall sets the line state to `OFF_HOOK` if any of the following conditions occur:

- The user-supplied function returns a value of 1.
- The final call progress result is one of the following:

FCP_BUSY1	301
FCP_BUSY2	302
FCP_ROBUSY	303
FCP_RECALL	304
FCP_PULSE	306
FCP_DIALTON	318
FCP_RNGNOANS	325
FCP_G2DETCT	326
FCP_SITINTC	327
FCP_QUIET	328
FCP_SITVACODE	329
FCP_SITREORD	330
FCP_SITNOCIR	331
FCP_CNGDETCT	332
FCP_UNKNOWN	340
FCP_ISDN_CALL_COLLISION	349

If the final call progress result is `FCP_HUMAN(316)`, `FCP_ANSWER(317)`, or `FCP_SILENCE(324)`, *BfvLineOriginateCall* sets the line state to:

Line State	Call_Protocol
OFF_HOOK	_FAX
CONNECTED	_VOICE

If the final call progress result is `FCP_ANSWER_TONE_DETECT (339)`, *BfvLineOriginateCall* sets the line state to:

Line State	Call_Protocol
CONNECTED	_FAX
CONNECTED	_VOICE

The user-supplied function can serve several purposes as follows:

- Report call progress values for debug functions
- Report intermediate call progress results (e.g., ring-back)
- Detect other call progress patterns not detected by firmware

The firmware automatically adapts to most international call progress signals.

If the application specifies the `CALL_PROTOCOL_RAW` value for *args.call_protocol_code*, the application must provide a function to analyze the raw call progress data.

The user-supplied function can access the call progress data using the *BfvDataCP* function. The user-supplied function must not call any function that causes a delay, such as waiting for a DTMF tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately.

For applications using Euro-ISDN on E1 or BRI lines (but not T1 ISDN lines), this function automatically sends a dial string using the protocol's overlapped dialing feature when the phone number exceeds 20 digits. The Euro-ISDN protocol only allows applications to send 20 digits in a block when placing a call. For phone numbers exceeding 20 digits, the protocol uses a process called overlapped dialing. This process sends extra digits after the initial call setup, allowing:

- The application to dial very large phone numbers
- The remote end to start answering a call before it receives all the digits

Users can place a call with up to 255 digits in the dial string. The Bfv API automatically breaks up the dial string into multiple blocks of 20 digits, and uses the overlapped dialing feature in the protocol to send one block of digits at a time. This process does not require any changes in the application.

Some protocols or locations do not support overlapped dialing. For example, T1 ISDN only allows a maximum of 24 digits.

If you use the low-level call control functions, use the *BfvCallSetup* and *BfvCallWaitForComplete* functions to perform the equivalent outbound calling process.

For an example of processing the 183 Session Progress headers during call setup, see the *faxll.c* sample application.

International Issues

In some countries, the PTT imposes certain dialing restrictions. Applications dialing a fax machine in any of these countries must use either the *BfvLineOrigCallDB* function or the *BfvDialDBCheck* function in conjunction with the *BfvDialDBUpdate* function. Otherwise, the application might be non-compliant with the target country's regulations.

For more information, see either:

- *Chapter , Dialing Database Functions on page 420*
- *Country-Specific Dialing Requirements on page 1450*

See Also

BfvCallSetup, *BfvCallWaitForComplete*, *BfvDataCP*, *BfvLineDialString*, *BfvLineOrigCallDB*

Example

```
BTLINE *lp;
struct args_telephone args_tel;
.
.
.

BT_ZERO(args_tel);
args_tel.call_protocol_code = CALL_PROTOCOL_FAX;
args_tel.phonenum = "9,w5551212";
BfvLineOriginateCall (lp, &args_tel);
```

BfvLineTerminateCall

Purpose

Sets the line state to on-hook and completes the disconnect process.

Syntax

```
void
BfvLineTerminateCall        (lp, args)
   BTLINE                   *lp;
   struct args_telephone    *args;
```

The structure contains the following fields.

Input Field

```
long timeout;
```

Output Fields

```
int cause_code;
int subcause;
int cause_location;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.timeout

Specifies an integer value that determines the length of time to wait for the function to complete. Valid values are:

0	Indicates waiting forever with no timeout.
nonzero	Indicates the number of milliseconds to wait for the function to complete.

Output

Return value: None.

`args.cause_code`

Returns an ISDN-defined code if the call fails that provides the reason why the function failed to finish disconnecting the call. See *Volume 6, Appendix D, Defining ISDN Cause Codes* for a description.

The value returned in this field only has significance if the application calls *BfvLineTerminateCall* for a line that is already idle.

`args.subcause`

Returns the response code of the SIP message that resulted in the termination of the call. This field is only applicable to calls using the SIP protocol. The definitions of these response codes are specified in RFC 3261.

`args.cause_location`

Returns an ISDN-defined code that indicates the originator (local or remote) of the failure notification (args.cause_code). For IP protocol calls (SIP or H.323), this field indicates which side initiated the termination of the call.

Valid values are:

IISDNlocUSERUser

IISDNlocPVT_LOCAL Private network serving the local user

IISDNlocPUB_LOCAL Public network serving the local user

IISDNlocTRANSIT_NET Transit network

IISDNlocPUB_REMOTE Public network serving the remote user

IISDNlocPVT_REMOTE Private network serving the remote user

IISDNlocINTERNATIONALInternational network

IISDNlocBEY_INTERWORKNetwork beyond internet-working point

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this function to end a call placed by the application or the remote end, or when an error occurs. *BfvLineTerminateCall* puts the call for the line in a disconnect or "on-hook" state.

Under IP call control, when using a channel to both receive and originate calls, applications should call the *BfvLineTerminateCall* function to disable incoming call detection before making an outbound call.

All calls must use *BfvLineTerminateCall* to properly complete the disconnect process. You can also use the equivalent low-level functions *BfvCallDisconnect* and *BfvCallWaitForRelease* to initiate and complete the disconnect process.

See Also

BfvCallDisconnect, *BfvCallReject*, *BfvCallWaitForRelease*

Example

```
BTLINE *lp;
char phonenum[32] = "w9w6175551234";
char local_id[20] = "1234567890abcdefg";
int calls_to_make; /* flag indicating we wish to end */
                   /* the call and make another */
struct args_telephone args_tel;
struct args_fax args_fax;

while (calls_to_make)
{
   BT_ZERO(args_tel);
   args_tel.call_protocol_code = CALL_PROTOCOL_FAX;
   args_tel.phonenum = phonenum;
   BfvLineOriginateCall(lp, &args_tel);

   /* Check returns, handle errors */

   BT_ZERO(args_fax);
   args_fax.s _ips = ips;
   args_fax.local_id = local_id;
   BfvFaxSend(lp, &args_fax);

   /* Check returns, handle errors */

   BT_ZERO(args_tel);
   BfvLineTerminateCall(lp, &args_tel);
}
```

BfvLineTransfer

Purpose

Automatically transfers an incoming call from the called party to the dialed transfer number, or returns control to the application so that it can determine whether to complete or cancel the transfer.

The SR140 and analog DID lines do not support this function.

Syntax

```
void
BfvLineTransfer              (lp, args)
   BTLINE                    *lp;
   struct args_telephone     *args;
```

The structure contains the following fields.

Input Fields

```
char *arg;
int call_mode;
int call_protocol_code;
int (*func)(BTLINE *lp, char *arg);
char *phonenum;
long timeout;
BTLINE *lp_second_channel;
int supervised;
int transfer_line_state;
int hold_call;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.arg

Provides an input argument for the *func* feature field. This field accepts a NULL setting if the user-defined function does not need an argument.

`args.call_mode`

Specifies a Boolean value that indicates whether the outbound call expects to receive a fax transmission when the connection completes. Set the field as follows:

`FALSE`

Indicates that the outbound call does not expect to receive a fax.

`TRUE`

Indicates that the outbound call expects to receive a fax after establishing the connection.

`args.call_protocol_code`

Selects a calling protocol for the channel from one of the following:

`CALL_PROTOCOL_FAX`

Selects the fax protocol. Requests the channel to report the results from the channel's call progress analysis along with the raw data. The channel reports the results as soon as it establishes the fax connection or encounters a busy condition.

Other detected call progress results are returned after the `ced_timeout` (the length of time to wait for the called stations's id signal) time-out.

`CALL_PROTOCOL_VOICE`

Selects the voice protocol. Requests the module to report the results from the channel's call progress analysis along with the raw data. The channel reports the results as soon as it detects a human or other answer condition.

`CALL_PROTOCOL_RAW`

Requests the channel to report the raw HIGH/LOW call progress results without performing any analysis.

`CALL_PROTOCOL_FAX_NO_RAW`
`CALL_PROTOCOL_VOICE_NO_RAW`

Selects either the fax or voice protocol, requesting the channel to report the results from the channel's call progress analysis without including the raw data.

`args.func`

Specifies a pointer to a user-supplied integer function that the channel calls during call progress. The channel calls this user function in a loop until it returns a final call progress result, or the user function indicates termination of call progress by returning 1.

Args.func is called as `(*args.func)(lp,args.arg)`. The *lp* argument contains the pointer to the line structure; the *args.arg* argument contains the supplied user-defined argument.

`args.phonenum`

Specifies a pointer to a null-terminated ASCII string that identifies the phone number to dial. Dial strings (phone numbers) may be up to a maximum of 255 characters (`ECC_MAX_DIGIT_STR` - 1).

PSTN Telephony

The dial string field supports the following digits and control characters. Invalid characters are ignored; upper and lower case letters are equivalent. Some protocols ignore control characters and only accept DTMF characters.

Valid Digits and Control Characters

0 - 9	Dials digits ‘0’ through ‘9’.
# (pound)	Dials a pound.
* (asterisk)	Dials an asterisk.
A - D	Sends the DTMF tone corresponding to the specified alphabetic character.
p	Changes the current or default dialing mode from tone dialing to pulse dialing.
t	Changes the current or default dialing mode from pulse dialing to tone dialing.
w	Waits for dial tone.
, (comma)	Causes a 1-second pause.
; (semicolon), i or I	Causes a 5-second pause. To create longer pauses, string any of these characters together.
! (exclamation point)	Sends a hook flash on analog and T1 robbed bit modules.

Note: In an analog environment or when using a T1 robbed bit FXS loopstart or E1 CAS loopstart protocol, the ‘w’ character means wait for dial tone. All other protocols ignore the ‘w’ and ‘i’ characters. Only analog environments and T1 robbed bit or E1 CAS protocols use the ‘p’, ‘t’, comma and semicolon characters.

`args.timeout`

Specifies an integer value that determines the length of time to wait for the function to complete. Valid values are:

`0`	Indicates waiting forever with no timeout.
`nonzero`	Indicates the number of milliseconds to wait for the function to complete.

`args.lp_second_channel`

A reference to the *lp* of the line to use for the enquiry call for protocols that require two B-channels to transfer a call (for example, Release Link Trunk (RLT) protocol).

Set to 0 when the transfer occurs over a single channel (for example, an analog line).

`args.supervised`

Specifies a Boolean value that determines whether the function automatically completes the transfer or returns without completing the transfer when it detects the value defined for the *args.transfer_line_state* argument. Valid values are:

`FALSE`

Indicates that the function automatically completes the transfer when it detects the value defined for *args.transfer_line_state*.

`TRUE`

Indicates that the function returns control to the application without completing the transfer when it detects the value defined for *args.transfer_line_state*.

`args.transfer_line_state`

Specifies the call state that determines when to complete the transfer. Valid values are:

`BST_DIAL_COMPLETE`

Completes the call transfer after dialing the transfer number.

`BST_ALERTING`

Completes the call transfer after detecting a ring.

`BST_CONNECTED`

Completes the call transfer when the called party answers.

`args.hold_call`

Specifies an integer value that determines whether to place the original call on hold or allow it to remain active. Use this field when transferring a call using two B channels; the field has no effect on single B-channel call transfers that must place the original call on hold to make the transfer. Valid values are:

`0`	Allows the original call to remain active and does not place it on hold before making the enquiry call on the other B channel.
`nonzero`	Places the original call on hold before making the enquiry call on the other B channel.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function performs the following to transfer an incoming call:

- Places the existing call on hold when the transfer occurs over a single channel.
- Dials the transfer number.
- Waits for the specified transfer state to be satisfied.

Then the function uses the value set in the *args.supervised* field as follows:

- When set to `TRUE`, returns control to the application allowing it to determine whether to complete or cancel the transfer, or
- When set to `FALSE`, automatically completes the call transfer and returns the original calling party's line state to idle.

When the application chooses to transfer a call without supervision, the completion of this function signals the end of the call and a return of the line state to idle.

When the application chooses to supervise the call transfer, the *BfvLineTransfer* function places the calling party on hold and connects the application to the transfer number. The function returns control to the application when it detects the line state the application selected to determine the timing. After the application regains control from the *BfvLineTransfer* function, it can choose to:

- Complete the transfer using the *BfvLineTransferComplete* function, or
- Cancel the transfer using the *BfvLineTransferCancel* function.

See Also

BfvLineTransferCancel, *BfvLineTransferComplete*

Example

```
BTLINE *lp;
struct args_telephone args_tel;
.
. /* Establish first call */
.

BT_ZERO(args_tel);
args_tel.call_protocol_code = CALL_PROTOCOL_FAX;
args_tel.phonenum = “123”;
args_tel.transfer_line_state = BST_CONNECTED;
BfvLineTransfer(lp, &args_tel);
if (args_tel.res.status != BT_STATUS_OK)
{
    /* Failed to transfer the call */
}
```

BfvLineTransferCancel

Purpose

Ends a previously initiated call transfer and retrieves the original calling party.

The SR140 and analog DID lines do not support this function.

Syntax

```
void
BfvLineTransferCancel        (lp, args)
   BTLINE                    *lp;
   struct args_telephone     *args;
```

The structure contains the following fields.

Input Field

```
long timeout;
BTLINE *lp_second_channel;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.timeout

Specifies an integer value that determines the length of time to wait for the function to complete. Valid values are:

0	Indicates waiting forever with no timeout.
nonzero	Indicates the number of milliseconds to wait for the function to complete.

args.lp_second_channel

A reference to the *lp* of the line to use for the enquiry call for protocols that require two B-channels to transfer a call (for example, Release Link Trunk (RLT) protocol).

Set to 0 when the transfer occurs over a single channel (for example, an analog line).

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this function to correctly cancel a call transfer that the application initiated with the *BfvLineTransfer* function. The *BfvLineTransferCancel* function drops the connection to the transfer party and returns control to the originating party.

See Also

BfvLineTransfer

Example

```
BTLINE *lp;
struct args_telephone args_tel;
.
.
.
/* Establish call and begin call transfer in
   supervised mode */
.
.
.
BT_ZERO(args_tel);
BfvLineTransferCancel(lp, &args_tel);
```

BfvLineTransferCapabilityQuery

Purpose

Retrieves information about the transfer capability of a channel.

Syntax

```
void
BfvLineTransferCapabilityQuery(lp, args)
   BTLINE                    *lp;
   struct args_telephone     *args;
```

The structure contains the following fields.

Input Field

None

Output Fields

```
RES res;
int transfer_mode;
int transfer_group;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing output fields.

Output

Return value: None.

args.transfer_mode

Returns a value that indicates how a line supports call transfers. Valid values are:

LINE_XFER_NONE

Line does not support call transfer.

LINE_XFER_SINGLE

Line only supports single B-channel (same channel) call transfer.

LINE_XFER_TWO_CHAN

Line only supports two B-channel (explicit) call transfer.

`LINE_XFER_ALL`

Line supports single and two B-channel call transfers.

`LINE_XFER_TWO_CHAN_NEEDS_NAILUP`

Line supports two B-channel call transfers, but the application must connect the B channels together (QSIG protocol requirement). When the application calls *BfvCallWaitTransferComplete* (see *page 359*), this function provides a feature to make the B-channel connection automatically. If the application has turned this feature off (set *args.disable_auto_sw_connect* to `TRUE`), the application must use the *BfvCallSwitchConnect* function to connect the two B channels together to make the call transfer.

`args.transfer_group`

Returns values to the application that indicate which lines can be paired to perform a two B-channel call transfer. Only lines that support two B-channel call transfers return valid values in this field.

This field returns matching line values for lines that can be paired to perform a two B-channel call transfer. When the line values do not match, the lines cannot be paired to perform a two B-channel call transfer.

Details

Use this function to retrieve information about the call transfer capability that a channel's protocol supports.

The function indicates that the call transfer feature is not supported for applications using an H.323 call control stack.

If the returned information indicates support for two B-channel call transfer, this function also provides the application with the information necessary to determine which two lines can be paired to perform two B-channel transfer.

Example

```
BTLINE *lp;
struct args_telephone args_tel;
.
.
.
BT_ZERO(args_tel);
BfvLineTransferCapabilityQuery(lp, &args_tel);
```

BfvLineTransferComplete

Purpose

Completes the call transfer connection for a previously initiated call transfer.

The SR140 and analog DID lines do not support this function.

Syntax

```
void
BfvLineTransferComplete     (lp, args)
   BTLINE                   *lp;
   struct args_telephone    *args;
```

The structure contains the following fields.

Input Field

```
long timeout;
BTLINE *lp_second_channel;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.timeout

Specifies an integer value that determines the length of time to wait for the function to complete. Valid values are:

0	Indicates waiting forever with no timeout.
nonzero	Indicates the number of milliseconds to wait for the function to complete.

`args.lp_second_channel`

A reference to the *lp* of the line to use for the enquiry call for protocols that require two B-channels to transfer a call (for example, Release Link Trunk (RLT) protocol).

Set to 0 when the transfer occurs over a single channel (for example, an analog line).

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this function to correctly finish a call transfer that the application initiated with the *BfvLineTransfer* function. The *BfvLineTransferComplete* function completes the connection to the transfer party and returns control to the originating party.

See Also

BfvLineTransfer

Example

```
BTLINE *lp;
struct args_telephone args_tel;
.
.
.
/* Establish call and begin call transfer in
   supervised mode */
.
.
.
BT_ZERO(args_tel);
BfvLineTransferComplete(lp, &args_tel);
```

BfvLineWaitForCall

Purpose

Activates a callback routine to wait for an incoming call and to perform call screening based on the routing information received with the call.

Syntax

```
void
BfvLineWaitForCall          (lp, args)
   BTLINE                   *lp;
   struct args_telephone    *args;
```

The structure contains the following fields.

Input Fields

```
long timeout;
char *phonenum;
unsigned sip_header_list_len;
BT_SIP_HEADER_LIST *sip_header_list;
```

Output Fields

```
CALL_RES call_res;
RES res;
enum TRxTransportType call_transport;
unsigned sip_header_list_len;
BT_SIP_HEADER_LIST *sip_header_list;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

`args.timeout`

Specifies an integer value that determines the length of time to wait for the function to complete. Valid values are:

`0`	Indicates waiting forever with no timeout.
`>0`	Indicates the number of milliseconds to wait for the function to complete.
`<0`	Indicates the number of milliseconds to wait for the call. However, if the function times out, it does not automatically reject calls that the system detects before the application makes the next call to *BfvLineWaitForCall*.

`args.phonenum`

Pointer to a null-terminated ASCII string that identifies the DID phone number the incoming call must match in order for the system to present the call to the application.

Only calls using an internet protocol (IP) such as SIP or H.323 can use this field. PSTN line types ignore the contents of this field.

`args.sip_header_list_len`

Specifies the size of the memory buffer pointed to by the *sip_header_list* field. This field should be set when the Bfv application wants to retrieve the SIP header information specified in an inbound SIP INVITE request used to establish a SIP call.

This field should be set to the size of the memory buffer pointed to by the *sip_header_list* field.

If the Bfv application doesn't want to retrieve the SIP header information specified in an inbound SIP INVITE request, this field should be set to 0 and *sip_header_list* should be set to NULL.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

`args.sip_header_list`

Specifies a pointer to a memory buffer allocated by the Bfv application to receive SIP header data from an inbound SIP INVITE request used to establish a SIP call. Included with the SIP header data will be the SIP Request-URI information. The Request-URI information will be the first item present in the SIP header data. The size of the memory buffer pointed to by *sip_header_list* should be specified in the *sip_header_list_len* field.

This field should be set to the address of a memory buffer allocated by the Bfv application that will receive the SIP header data. As *sip_header_list* is a pointer to a *BT_SIP_HEADER_LIST* structure, the Bfv application memory buffer should be allocated and initialized in a manner similar to the following:

```
args.sip_header_list =
      (BT_SIP_HEADER_LIST *)malloc(1000);
memset(args.sip_header_list, 0, 1000);
args.sip_header_list_len = 1000;
```

If the Bfv application doesn't want to retrieve the SIP header information specified in an inbound SIP INVITE request, *sip_header_list_len* should be set to 0 and *sip_header_list* should be set to NULL.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

Note: The application must free this structure after use to release the memory.

Output

Return value: None.

`args.call_res`

A structure containing information about the incoming call, including call type and DID digits. For a detailed description of the `CALL_RES` structure parameters, see *Volume 6, Appendix B, CALL_RES Structure Parameters.*

When the call control configuration file (*callctrl.cfg*) has the *caller_id* parameter enabled for an analog port, the Bfv API returns the name of the caller in the *calling_party_subaddress* and the *name_ident* fields of the `CALL_RES` structure. The Bfv API returns the caller ID information for an E1 or T1 QSIG port in the *name_ident* and *name_char_set* fields of the `CALL_RES` structure.

When receiving a diverted incoming call on a port using the QSIG protocol, this field outputs the *`redir_number`* and *`redir_reason`* fields of the `CALL_RES` structure to indicate the phone number of the device diverting the call and the reason for diverting the call. Valid diversion reasons are:

`DIVERT_NONE`	Used for call that does not divert.
`DIVERT_BUSY`	Call diverted for busy condition.
`DIVERT_UNCONDITIONAL`	Call diverted without conditions.
`DIVERT_NO_RESPONSE`	Call diverted for unresponsive line.

`args.call_transport`

Returns a code indicating the transport protocol used for the inbound call. Valid values are :

TRANSPORT_TYPE_UDP

User Datagram Protocol (UDP) was the transport protocol used for the inbound SIP call.

TRANSPORT_TYPE_TCP

Transmission Control Protocol (TCP) was the transport protocol used for the inbound SIP call.

TRANSPORT_TYPE_DEFAULT

This code is returned for calls using the H.323 internet protocol and PSTN line types.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

`args.sip_header_list_len`

If a SIP call has successfully been received, this field will be set to the amount of data in the sip_header_list memory buffer that has been populated with the SIP Request-URI data followed by the SIP header data. If the memory buffer pointed to by *sip_header_list* isn't large enough to hold all the SIP header data from the inbound SIP INVITE request, then *sip_header_list_len* will be set to a value of *SIP_HEADER_INVALID_LEN*. In this case, the memory buffer pointed to by *sip_header_list* will only be populated with the SIP Request-URI data followed by the complete SIP header name/header value pairs that fit in the buffer.

For example, if there are 10 SIP headers in the inbound SIP INVITE request, but the memory buffer pointed to by *sip_header_list* after the SIP Request-URI data can only hold enough data for 9 complete SIP header name/header value pairs retrieved from the SIP INVITE request with 30 bytes of the buffer unused, then *sip_header_list_len* will be set to a value of *SIP_HEADER_INVALID_LEN* and the SIP Request-URI data followed by only 9 complete SIP header name/header value pairs will be returned in the *sip_header_list* buffer. No partial or incomplete SIP header information from the 10th header will be used to populate the remaining 30 bytes in the *sip_header_list* buffer.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

`args.sip_header_list`

If a SIP call has successfully been received, the memory buffer pointed to by *sip_header_list* will be populated with a singly linked list of SIP header data stored in *BT_SIP_HEADER_NODE* structures starting with a *BT_SIP_HEADER_LIST* structure.

The data structures used to access the SIP header data in the singly linked list are defined below:

```
struct BT_SIP_HEADER {
    char *header_name;
    char *header_value;
};
typedef struct _BT_SIP_HEADER_NODE {
    struct BT_SIP_HEADER header;
    struct _BT_SIP_HEADER_NODE *next_header;
} BT_SIP_HEADER_NODE;
typedef struct _BT_SIP_HEADER_LIST {
    int num_sip_headers;
    BT_SIP_HEADER_NODE sip_headers;
} BT_SIP_HEADER_LIST;
```

Some SIP headers that have multiple values may be returned as several SIP headers. For example, the following header:

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

Will be returned as five separate headers:

Header Name	*Header Value*
Allow	INVITE
Allow	ACK
Allow	OPTIONS
Allow	CANCEL
Allow	BYE

If some of the SIP header names in the inbound SIP INVITE request were specified in compact form, they may be returned to the Bfv application in long form (e.g., Content-Type header name specified in inbound SIP request as "c", returned to Bfv application as "Content-Type"). The SIP URI-Request name will be returned to the application as "Request-URI".

The maximum supported number of SIP headers that can be returned is 98.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

Details

Use this function to wait for detection of an incoming call and allow your application to screen the call based on the routing information it receives with the call. The function waits until an incoming call has been detected or the timeout expires.

An application can use the *args.phonenum* input field to implement basic inbound call routing for an IP-enabled application. Unlike applications for PSTN line types, any channel in the system can answer incoming IP calls. The *args.phonenum* field allows the application to specify a DID number that an incoming call must match before the system presents the call to the application. The Bfv API only attempts to match the number of digits passed into this function in this pointer. For example, if you specify 4 digits to pass in, the Bfv API only seeks to match the first 4 digits of the incoming call, and it ignores any extra digits in the incoming call's DID string. This field only works for incoming calls using SIP and H.323 internet protocols. The Bfv API ignores the field for any incoming call on PSTN line types.

If no error occurs when the function executes, *args.res.status* contains `BT_STATUS_OK`.

If a call is:

- Detected, the *args.res.line_status* field contains `WAIT_FOR_CALL_OK`.
- Not detected within the timeout, the *args.res.line_status* field contains `WAIT_FOR_CALL_TIMEOUT`.

An early return occurs when *timeout* is a nonzero value and the specified time interval expires.

For DID lines, *BfvLineWaitForCall* waits to capture DID digits. See *Volume 6, Appendix A, Call Control Configuration File* to define the DID digit parameters.

Applications can execute call screening (for example, on DID digits) with the *BfvLineAnswer* and *BfvLineTerminateCall* functions. If the telephone number is invalid, the application can terminate the call with the *BfvLineTerminateCall* or *BfvCallReject* functions.

The call control configuration file (*callctrl.cfg*) stores the configuration for DID channels (the expected number of DID digits to receive [`max_did_digits`]).

Using a negative value for *args.timeout* permits an application to begin ring detection and continue it during the interval between calls to this function. This capability permits a program to repeatedly switch, at short intervals, between waiting for a ring and processing other tasks without risking any loss of rings due to timing between enabling/disabling ring detection and the actual ring occurrence.

If the application does not detect a ring and decides to do something else (for example, make a phone call), the application must disable ring detection. To disable ring detection:

- Use *BfvLineWaitForCall* and set *args.timeout* to a small positive value (1)

  or

- Use the *BfvCallRingDetect* function and set `args.mode = 0`

See Also

BfvCallRingDetect, *BfvCallWaitForSetup*, *BfvLineAlert*, *BfvLineAnswer*, *BfvLineTerminateCall*

Example

```
BTLINE *lp;
struct args_telephone args;

BT_ZERO(args);
args.timeout = 0L; /* wait until incoming call occurs */
   /* Use next line example only to implement call routing
      for an IP-enabled application */
args.phonenum = "4083700881";
BfvLineWaitForCall(lp, &args);
if (args.res.line_status != WAIT_FOR_CALL_OK)
{
    printf("Error While Waiting for Call\n");
}
```

BfvLoopCurrentDetectDisable

Purpose

Turns off loop current detection for the specified channel.

Syntax

```
void
BfvLoopCurrentDetectDisable(lp, args)
   BTLINE                    *lp;
   struct args_telephone     *args;
```

The structure contains the following fields.

Input Field

None

Output Fields

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this function to turn off loop current detection for the specified channel. If loop current detection is already turned off, calling this function has no effect and does not return an error value.

Turning off loop current detection prevents the system from immediately terminating a fax or voice function call and returning the BT_STATUS_ERROR_HANGUP `res.status` value. When the far end of a call disconnects with loop current detection turned off:

- fax functions terminate with a status code set
- voice functions might or might not terminate with a status code set, depending on the settings turned on for voice functions.

You must call this function after the call has been established and before calling any fax or voice functions.

If your application uses call transfer or call retrieve functions, loop current detection is automatically enabled when the transfer or retrieve function terminates.

Do not call this function while a fax or voice function is in progress. Doing so might cause a failure of the fax or voice function to start or stop properly, or result in a loss of data.

Note: Functions like *BfvLineOriginateCall* and *BfvLineWaitForCall* automatically turn on loop current detection.

See Also

BfvLoopCurrentDetectEnable

Example

```
BTLINE *lp;
struct args_telephone args;
.
.
.
BT_ZERO(args);
BfvLoopCurrentDetectDisable(lp, &args);
```

BfvLoopCurrentDetectEnable

Purpose

Turns on loop current detection for the specified channel.

Syntax

```
void
BfvLoopCurrentDetectEnable (lp, args)
   BTLINE                   *lp;
   struct args_telephone    *args;
```

The structure contains the following fields.

Input Field

None

Output Fields

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this function to turn on loop current detection for the specified channel if the application previously turned off loop current detection (*BfvLoopCurrentDetectDisable*). Since the release of Brooktrout SDK 3.0, the Bfv API always enables loop current detection at the start of a call. If loop current detection is already turned on, calling this function has no effect and does not return an error value.

You must call this function after the call has been established and before calling any fax or voice functions.

Do not call this function while a fax or voice function is in progress. Doing so causes a failure of the fax or voice function to start or stop properly, or results in a loss of data.

Note: Functions like *BfvLineOriginateCall* and *BfvLineWaitForCall* automatically turn on loop current detection.

See Also

BfvLoopCurrentDetectDisable

Example

```
BTLINE *lp;
struct args_telephone args;
.
.
.
BT_ZERO(args);
BfvLoopCurrentDetectEnable(lp, &args);
```

11 - Dialing Database Functions

This chapter describes the functions an application uses to dial facsimile machines from a database.

It has the following sections:

- *Dialing Database Function Call Summary on page 422*
- *Dialing Database functions listed alphabetically*

Many countries have dialing restrictions that require applications to maintain a database of information about previously called fax telephone numbers. If the application dials a fax machine in one of these countries, it must use the dialing database functions to avoid violating the target country's PTT regulations.

Use the dialing database functions only if your application sends facsimiles into countries with such restrictions. For a list of the countries the API supports and a description of their dialing restrictions, see *Volume 6, Appendix G, Country-Specific Parameter Files*.

An application uses the dialing database functions to send a fax only when it expects a fax machine to answer. An application must not use the dialing database functions when it expects a human or a voice answering machine to answer.

An application uses the dialing database locally and only on a single computer. It must not share the dialing database with multiple computers over a network. The API does not provide for sharing or for format differences in intercompiler time storage.

The dialing database functions use a lock file to ensure exclusive access to the database. The name of this file is *btdb.lck*, and it resides in the same directory as the dialing database. If execution of the program stops prematurely (that is, the system crashes), the lock file might remain. If this occurs, you must manually remove the lock file so the dialing database functions can proceed.

The API uses the C library `time()` function for timing purposes. Some libraries implement this function slightly differently. Users must make sure that all programs using the *DialDB* functions on a particular computer are linked with the same version of their compiler library.

See *Volume 6, Appendix G, Country-Specific Parameter Files* for more information about implementing dialing restrictions.

Dialing Database Function Call Summary

The dialing database functions include the following shown in *Table 16*.

Table 16. Dialing Database Function Summary

Function	Purpose	Page
BfvDialDBCheck	Checks the specified dialing database for the specified telephone number and returns the amount of time the application must wait before dialing the telephone number.	*423*
BfvDialDBList	Enables the application to read the contents of the specified dialing database.	*425*
BfvDialDBUpdate	Updates the specified dialing database with the results of the most recent call to the specified telephone number.	*428*
BfvLineOrigCallDB	Checks the specified dialing database for the specified telephone number, returns the amount of time to wait before dialing, and then places the call on an outgoing line and updates the dialing database.	*431*

BfvDialDBCheck

Purpose

Checks the specified dialing database for the specified telephone number and returns the amount of time the application must wait before dialing the telephone number.

Syntax

```
int
BfvDialDBCheck              (lp, args)
   BTLINE                   *lp;
   struct args_telephone    *args;
```

The structure contains the following fields.

Input Fields

```
char *dbfile;
char *raw_number;
```

Output Fields

```
long wait_time;
int reason;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.db_file

Name of the dialing database file.

args.raw_number

String containing the telephone number to dial. This string must contain the actual telephone number only; no long distance, trunk, or other codes are valid. All characters except digits, # (pound or number symbol), and * (star or asterisk) will be ignored.

Output

Return value:

0 Successful database lookup.

–1 Error accessing database file.

`args.wait_time`

The amount of time to wait before dialing, in seconds.

`args.reason`

The *reason* for the delay specified by *args.wait_time*.

The reason will be one of the `DL_REASON_...` values defined in *dialdb.h*.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

The application must call this function before it calls *BfvLineOriginateCall*. If this function returns a nonzero *args.wait_time* value, the application must not proceed with dialing the telephone number. After calling *BfvLineOriginateCall*, the application must call *BfvDialDBUpdate* to update the dialing database. The application can use the *BfvDialDBList* to examine the contents of the dialing database.

Checks the dialing database specified by *args.db_file* for information about *args.raw_number*. Using *args.wait_time*, it returns the amount of time the application must wait before it dials *args.raw_number* on channel *lp*. It returns the reason for the delay using *args.reason*.

See Also

BfvDialDBList, *BfvDialDBUpdate*, *BfvDialDBUpdate*

Example

See the *dlfax.c* application in the sample applications directory.

BfvDialDBList

Purpose

Enables the application to read the contents of the specified dialing database.

Syntax

```
int
BfvDialDBList                (args)
   struct args_telephone    *args;
```

The structure contains the following fields.

Input Fields

```
char *db_file;
int (*db_func)(struct dialdb *dbentry, char *arg);
char *arg;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.db_file

Name of the dialing database file.

args.db_func

Pointer to a user-supplied integer function that is called once for each entry in the dialing database.

The value of *args.db_func* cannot be `NULL`. The function *args.db_func* is called as `(*args.db_func)(dbentry,args.arg)`, where `dbentry` is a **`struct dialdb`** * pointer to a structure that describes a database entry, and `arg` is the user-supplied argument.

If *args.db_func* returns 0, processing of the database continues. Otherwise, processing halts.

args.arg

A user-supplied argument for *args.db_func*, which can be `NULL`.

Output

Return value:

0 Successful database listing.

1 Error opening the database file.

2 Error reading the database file.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The application can call this function at any time. Before calling *BfvLineOriginateCall*, the application must call *BfvDialDBCheck*. After calling *BfvLineOriginateCall*, the application must call *BfvDialDBUpdate*.

For each database entry, it calls the function *args.db_func* with a *struct dialdb* * pointer and the user-supplied argument *args.arg*.

The dialing database file can often contain empty entries, which do not contain any database information and should be skipped. These entries have an empty telephone number; that is, the first byte of *phonenum* is 0.

Many countries have dialing restrictions that require applications to maintain a database of information about previously called fax telephone numbers. If the application dials a fax machine in one of these countries, it must use the dialing database functions to avoid violating the target country's PTT regulations. Use the dialing database functions only if your application sends facsimiles into countries with such restrictions. For a list of the countries the API supports and a description of their dialing restrictions, see *Volume 6, Appendix G, Country-Specific Parameter Files.*

An application uses the dialing database functions to send a fax only when it expects a fax machine to answer. An application must not use the dialing database functions when it expects a human or a voice answering machine to answer.

An application uses the dialing database locally and only on a single computer. The application must not share the dialing database with multiple computers over a network. The API does not provide for sharing or for format differences in intercompiler time storage.

The dialing database functions use a lock file to ensure exclusive access to the database. The name of this file is *btdb.lck*, and it resides in the same directory as the dialing database. If execution of the program stops prematurely (that is, the system crashes), the lock file might remain. If this occurs, you must manually remove the lock file so the dialing database functions can proceed.

The API uses the C library `time()` function for timing purposes. Some libraries implement this function slightly differently. Users must make sure that all programs using the *BfvDialDB* functions on a particular computer are linked with the same version of their compiler library.

See *Volume 6, Appendix G, Country-Specific Parameter File* for more information about implementing dialing restrictions.

See Also

BfvDialDBCheck, *BfvDialDBUpdate*, *BfvLineOrigCallDB*

Example

See the *dlfax.c* application in the sample applications directory.

BfvDialDBUpdate

Purpose

Updates the specified dialing database with the results of the most recent call to the specified telephone number.

Syntax

```
int
BfvDialDBUpdate              (lp, args)
   BTLINE                    *lp;
   struct args_telephone     *args;
```

The structure contains the following fields.

Input Fields

```
char *db_file;
char *raw_number;
RES *db_res;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.db_file

Name of the dialing database file.

args.raw_number

String containing the dialed telephone number. This string must contain the actual telephone number only; no long distance, trunk, or other codes are valid. All characters except digits, # (pound or number symbol), and * (star or asterisk) will be ignored.

arg.db_res

Pointer to a **RES** structure that contains the result of the last call attempt to *args.raw_number*.

This RES structure is usually the one contained within the *args_telephone* structure that *BfvLineOriginateCall* updates.

Output

Return value:

0 Successful database update.

–1 Error accessing the database file.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

The application must call this function after it calls *BfvLineOriginateCall*, whether *BfvLineOriginateCall* returns a successful call result or not. Before calling *BfvLineOriginateCall*, the application must call *BfvDialDBCheck*. The application can use the *BfvDialDBList* function to examine the contents of the dialing database.

To manually blacklist a telephone number (preventing applications from dialing it), call the *BfvDialDBUpdate* function with an *args.res* value of NULL. To manually remove a telephone number from the blacklist (permitting applications to dial it), call this function with `res.status = BT_STATUS_OK` and `res.line_status = FCP_ANSWER_TONE_DETECT` (the values that normally indicate a successful connection).

Many countries have dialing restrictions that require applications to maintain a database of information about previously called fax telephone numbers. If the application dials a fax machine in one of these countries, it must use the dialing database functions to avoid violating the target country's PTT regulations. Use the dialing database functions only if your application sends facsimiles into countries with such restrictions. For a list of the countries the API supports and a description of their dialing restrictions, see *Volume 6, Appendix G, Country-Specific Parameter Files.*

An application uses the dialing database functions to send a fax only when it expects a fax machine to answer. An application must not use the dialing database functions when it expects a human or a voice answering machine to answer.

An application uses the dialing database locally and only on a single computer. It must not share the dialing database with multiple computers over a network. The API does not provide for sharing or for format differences in intercompiler time storage.

The dialing database functions use a lock file to ensure exclusive access to the database. The name of this file is *btdb.lck*, and it resides in the same directory as the dialing database. If execution of the program stops prematurely (that is, the system crashes), the lock file might remain. If this occurs, you must manually remove the lock file so the dialing database functions can proceed.

The API uses the C library `time()` function for timing purposes. Some libraries implement this function slightly differently. Users must make sure that all programs using the *DialDB* functions on a particular computer are linked with the same version of their compiler library.

See *Volume 6, Appendix G, Country-Specific Parameter Files* for more information about implementing dialing restrictions.

See Also

BfvDialDBCheck, *BfvDialDBList*, *BfvLineOrigCallDB*

Example

See the *dlfax.c* application in the sample applications directory.

BfvLineOrigCallDB

Purpose

Checks the specified dialing database for the specified telephone number, returns the amount of time to wait before dialing, and then places the call on an outgoing line and updates the dialing database.

Syntax

```
void
BfvLineOrigCallDB (lp, args)
   BTLINE                    *lp;
   struct args_telephone     *args;
```

The structure contains the following fields.

Input Fields

```
char *phonenum;
int call_protocol_code;
int (* func) (BTLINE *lp, char *arg);
char *arg;
char *db_file;
char *raw_number;
unsigned report_cadence;
unsigned report_freq;
unsigned freq_report_time;
```

Output Fields

```
long wait_time;
int reason;
RES res;
```

Modified Fields

```
db_res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.phonenum

Specifies a pointer to a null-terminated ASCII string that identifies the phone number to dial. Dial strings (phone numbers) may be up to a maximum of 255 characters (`ECC_MAX_DIGIT_STR` - 1).

PSTN Telephony

The dial string field supports the following digits and control characters. Invalid characters are ignored; upper and lower case letters are equivalent. Some protocols ignore control characters and only accept DTMF characters.

Valid Digits and Control Characters

0 - 9	Dials digits ‘0’ through ‘9’.
# (pound)	Dials a pound.
* (asterisk)	Dials an asterisk.
A - D	Sends the DTMF tone corresponding to the specified alphabetic character.
p	Changes the current or default dialing mode from tone dialing to pulse dialing.
t	Changes the current or default dialing mode from pulse dialing to tone dialing.
w	Waits for dial tone.
, (comma)	Causes a 1-second pause.
; (semicolon), i or I	Causes a 5-second pause. To create longer pauses, string any of these characters together.
! (exclamation point)	Sends a hook flash on analog and T1 robbed bit modules.

Note: In an analog environment or when using a T1 robbed bit FXS loopstart or E1 CAS loopstart protocol, the ‘w’ character means wait for dial tone. All other protocols ignore the ‘w’ and ‘i’ characters. Only analog environments and T1 robbed bit or E1 CAS protocols use the ‘p’, ‘t’, comma and semicolon characters.

IP Telephony

For IP outbound calls using the H.323 protocol:

- `Phone#@IP Address:Port#`

  If the receiving side does not require a phone number, a value for `Phone#` is optional. Also, `:Port#` is optional — the Bfv API uses 1720 as the default port value.

- `TA:IP Address:Port#,Phone#`

  `:Port#` is optional — the Bfv API uses 1720 as the default port value. If the receiving side does not require a phone number, a value for `Phone#` is optional.

- `Name:<name of person to dial>`

  Use this form only if using gatekeeper

- `E164alias:7894561234`

  Use this form only if using gatekeeper

Note: DNS lookups are not supported in H.323. You can use an H.323 or E.164 alias in conjunction with a gatekeeper to provide similar functionality.

Examples

```
4082345555@10.155.89.6:175
4082345555@10.155.89.6

TA:10.155.89.6:175,4082345555
TA:10.155.89.6,4082345555
TA:10.155.89.6

Name:Fred Smith
E164alias:4082345555
```

For IP outbound calls using the SIP protocol:

`Phone#@IP Address:Port#`

If the receiving side does not require a phone number, a value for Phone# is optional. Also, :Port# is optional - the Bfv API uses 5060 as the default port value.

Examples (IPv4 Addresses)

```
4082345555@10.155.89.7:175
4082345555@10.155.89.7
```

Examples (IPv6 Addresses)

```
4082345555@[2000::2ef3:1dff:ea3]:175
4082345555@[fe80::1f4:189c:74da:69f7]
```

Note: IPv6 addresses must be enclosed in brackets. In addition, if a link-local IPv6 address is specified, the Scope ID should be omitted from the address.

DTMF Post Dialing

For all types of IP calls, the character '&' (ampersand) may be included to initiate post-dialing. This character indicates that the rest of phonenum specifies a sequence of DTMF digits to be "post-dialed" after messages from the remote side indicate the call is proceeding towards connecting.

Within the post-dial string, all dialing characters listed for PSTN Telephony are allowed except for 'p', 't', 'w', and '!'. The appearance of an additional '&' will terminate processing of the string.

Post-dialing of the specified digits will occur upon the first receipt of one of the following IP call control messages:

- SIP -- 183 Progress
- SIP -- 200 OK
- H.323 -- Progress
- H.323 -- Connect

The post-dial feature is controlled by the user configuration file parameter *post_dialing_enable*. If the feature is disabled by that parameter, then the ampersand has no special effect and the entire phonenum field is used as is.

Note: If a program accepts a phone number on the command line, the phone number will likely need to be quoted if it contains an ampersand, since this is a special character on most OSes. (Use double quotes (") on Windows, or single (') or double (") quotes on Linux.)

`args.call_protocol_code`

Selects a calling protocol for the channel. Supports:

`CALL_PROTOCOL_VOICE`

`CALL_PROTOCOL_FAX`

`CALL_PROTOCOL_RAW`

`CALL_PROTOCOL_VOICE_NO_RAW`

`CALL_PROTOCOL_FAX_NO_RAW`

Voice protocol – reports the results from the channel call progress analysis and the raw data; generates a quick return (that is, reports call progress results as soon as they are detected) when a human or other answer condition is detected.

Fax protocol – reports the results from the channel call progress analysis and the raw data; generates a quick return only when it has established a fax connection or encountered a busy signal.

Other detected call progress results are returned after the `ced_timeout` (the length of time to wait for the called station's ID signal).

The *BT_CPARM.CFG* or the user-defined configuration file determine the `wait_for_ced` value.

Raw protocol – reports raw HIGH/LOW call progress values without performing any analysis.

`args.func`

Pointer to a user-supplied integer function that is called during call progress. This function is called in a loop until a final call progress result is returned or the user function indicates termination of call progress.

Args.func is called as `(*args.func)(lp,args.arg)`. The *lp* argument contains the pointer to the line structure; the *args.arg* argument contains the supplied user-defined argument.

`args.arg`

An argument for the *args.func* feature. Can be `NULL`.

`args.db_file`

Name of the dialing database file.

`args.raw_number`

String containing the telephone number to dial. This string must contain the actual telephone number only; no long distance, trunk, or other codes are valid. All characters except digits, # (pound or number symbol), and * (star or asterisk) will be ignored.

`args.report_cadence`

If nonzero, enables cadence analysis and reporting. This mode is available in full call progress mode only, not in limited call progress mode (DISS).

Do not enable call progress programming in conjunctions with this option. (See BfvLineCallProgressProgram).

Use this argument for development and debugging purposes. When enabled, most other call progress analysis is disabled.

`args.report_freq`

If nonzero, enables frequency analysis and reporting. This mode is available in full call progress mode only, not in limited call progress mode (DISS).

Do not enable call progress programming in conjunction with this option. (See BfvLineProgressProgram).

Use this argument for development and debugging purposes. When enabled, most other call progress analysis is disabled.

`args.freq_report_time`

If nonzero, specifies frequency reporting interval, in milliseconds. The default is 1000.

Output

Return value: None.

`args.wait_time`

The amount of time to wait before dialing.

`args.reason`

The reason for the delay specified by *`args.wait_time`*.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function takes the place of *BfvDialDBCheck*, *BfvLineOriginateCall*, and *BfvDialDBUpdate*. The application can use the *BfvDialDBList* function to examine the contents of the dialing database.

This function can return `res.status = BT_STATUS_ERROR` and `res.line_status = APIERR_TOO_SOON`.

Many countries have dialing restrictions that require applications to maintain a database of information about previously called fax telephone numbers. If the application dials a fax machine in one of these countries, it must use the dialing database functions to avoid violating the target country's PTT regulations. Use the dialing database functions only if your application sends facsimiles into countries with such restrictions. For a list of the countries the API supports and a description of their dialing restrictions, see *Volume 6, Appendix G, Country-Specific Parameter Files.*

An application uses the dialing database functions to send a fax only when it expects a fax machine to answer. An application must not use the dialing database functions when it expects a human or a voice answering machine to answer.

The dialing database functions use a lock file to ensure exclusive access to the database. The name of this file is *btdb.lck*, and it resides in the same directory as the dialing database. If execution of the program stops prematurely (that is, the system crashes), the lock file might remain. If this occurs, you must manually remove the lock file so the dialing database functions can proceed.

The API uses the C library `time()` function for timing purposes. Some libraries implement this function slightly differently. Users must make sure that all programs using the *DialDB* functions on a particular computer are linked with the same version of their compiler library.

See *Volume 6, Appendix G, Country-Specific Parameter Files* for more information about implementing dialing restrictions.

See Also

BfvDialDBCheck, *BfvDialDBList*, *BfvDialDBUpdate*, *BfvLineDialString*, *BfvLineOriginateCall*

Example

See the *dlfax.c* application in the sample application directory.

12 - Data Structures

This chapter describes the data structures and a macro that an application uses to pass call control information to the Bfv API-level functions.

It has the following sections:

- *Low-Level Call Control (args_cc)*
- *Functions Using the args_cc Structure*
- *High-Level Call Control (args_telephone)*
- *Macros*

The following pages describe the contents of data structures that provide input or output to the call control functions. For call control functions, use one of two common structures depending on the level of control the application requires. These structures are:

- *Low-Level Call Control (args_cc) on page 439*

  A common structure providing input and output information for the low-level call control functions.
- *High-Level Call Control (args_telephone) on page 468*

  A common structure providing input and output information for high-level call control functions.

Low-Level Call Control (args_cc)

The low-level call control functions use the *args_cc* data structure to provide specific call control parameters for the called function.

This release of the Bfv API modifies this structure, removing fields that allow the user to pass module and port-specific call control configuration parameters to the system by calling the *BfvCallCtrlInit* function. You must use the configuration tool provided with your Brooktrout SDK (Windows systems), or edit your call control configuration (*callctrl.cfg*) file to make changes to module or port-specific call control parameters. You should use the **BfvLineReset** function to load and initialize new call control configuration parameters.

Table 17 through *Table 22*, starting on *page 466* show how functions use the fields in the *args_cc* structure. The structure contains the following fields:

```
struct args_cc
{
    int mode;
    int call_protocol_code;
    int cause;
    long timeout;
    const char *calling_party;
    char orig_called_num[ECC_MAX_DIGIT_STR];
    unsigned char calling_party_presentation;
    unsigned char calling_party_screening;
    enum TRxECCCallType call_type
    int call_mode;
    const char *phonenum;
    int state;
    RES res;
    CALL_RES cres;
    M_CB1 int (* M_CB2 db_func)(MILL_LINE *lp,char *arg);
    char *arg;
    char *btcall_file;
    void *reserved0;      /* Removed - TRxModuleInfo
                             structure */
    int *reserved1;       /* Removed - Number of elements
                             in structure */
```

```
    int  set_log_file;
    const char *log_file;
    MILL_LINE *lp_second_channel;
    int enquiry_call;
    int calls_on_hold;
    int ie_count;
    int ie_length;
    unsigned char ie_data;
    int cause_location;
    char name_ident[MAX_NAME_STR];
    int name_char_set;
    int disable_auto_sw_connect;
    CONNECTED_NUM connected_num;
    unsigned char override_calling_plan;
    unsigned char override_calling_type;
    int subcause;
    enum TRxTransportType call_transport;
    unsigned num_user_sip_headers;
    struct BT_USER_SIP_HEADER *user_sip_headers;
    int fax_media_feature_tag;
    unsigned fallback_rtp_reinvite;
    unsigned sip_header_list_len;
    BT_SIP_HEADER_LIST *sip_header_list;
    unsigned char override_called_plan;
    unsigned char override_called_type;
};
```

Fields in the args_cc Data Structure

Field	Description
mode	Specifies a value that determines how and whether the *BfvCallRingDetect* function detects incoming ring signals. Set this field as follows: `0` Turns ring detection behavior and static ring detection mode off. `-1` Turns static ring detection mode on. `1` Turns ring detection behavior on.

call_protocol_code	Specifies a calling protocol for the module that determines how and when the module reports the results of call progress analysis. Set the value of this field to:
	`CALL_PROTOCOL_FAX`
	Selects the fax protocol. This setting requests the module to report the results from the module's call progress analysis along with the raw data. The module reports the results as soon as it establishes the fax connection or encounters a busy condition.
	`CALL_PROTOCOL_RAW`
	Selects a value that requests the module to report the raw high and low call progress results without performing any analysis.
	`CALL_PROTOCOL_VOICE`
	Selects the voice protocol. This setting requests the module to report the results from the module's call progress analysis along with the raw data. The module reports the results as soon as it detects a human or other answer condition.
	`CALL_PROTOCOL_FAX_NO_RAW`
	`CALL_PROTOCOL_VOICE_NO_RAW`
	Selects either the fax or voice protocol, requesting the module to report the results from the module's call progress analysis without including the raw data.

cause	Specifies or returns an ISDN-defined code that provides the reason why the call failed. See *Volume 6, Appendix D, Defining ISDN Cause Codes*, for a description.
timeout	Specifies an integer value that determines the length of time to wait for the call control function to complete. Values are: `0` Indicates waiting forever with no timeout. `nonzero` Indicates the number of milliseconds to wait for the function to complete.

calling_party

Specifies a pointer reference to calling party information to send to the remote end during an outbound call. The field allows a maximum of 255 characters (`ECC_MAX_DIGIT_STR` - 1).

orig_called_num

Returns the number of the first destination for the outbound call. The field allows a maximum of 255 characters (`ECC_MAX_DIGIT_STR` - 1).

calling_party_presentation

Specifies or returns a value for ISDN calls only that indicates the origin of the calling party number presented to the called party or the accessibility level of the calling party number. Values are:

`ECC_PRES_APP_DEFINED`

Indicates that the application defined the presentation intention to the called party.

`ECC_PRES_ALLOWED`

Indicates that the network allows presentation of the calling party number to the called party.

`ECC_PRES_RESTRICTED`

Indicates that the network restricts presentation of the calling party number to the called party.

`ECC_PRES_NUM_NOT_AVAIL`

Indicates that the network does not have a calling party number specified to present to the called party.

calling_party_screening

Specifies or returns a value for ISDN calls only that indicates the origin and validity of the calling party number passed to the called party. Values are:

`ECC_SCRN_APP_DEFINED`

Indicates that the application provided the calling party number and defined its validation level.

`ECC_SCRN_USER_NOT_SCREENED`

Indicates that the network provided the calling party number without validating it.

`ECC_SCRN_USER_VERIFICATION_PASSED`

Indicates that the network provided the calling party number and passed a successfully validated number to the called party.

`ECC_SCRN_USER_VERIFICATION_FAILED`

Indicates that the network failed to validate the calling party number.

`ECC_SCRN_NETWORK_PROVIDED`

Indicates that the network validated the calling party number.

call_type

Specifies the call type to use when making the outbound call. Set a value in this field only if the ISDN port uses a BRI or PRI protocol. If you set the field for any other type of ISDN protocol, the system ignores this value. Use one of the following values for this field:

`ECC_CALL_TYPE_DEFAULT`

Makes the call using the default setting from the *callctrl.cfg* file.

`ECC_CALL_TYPE_VOICE`

Makes a voice call.

`ECC_CALL_TYPE_MODEM`

Makes a modem (3.1 kHz audio) call. This setting provides higher quality audio for the call.

`ECC_CALL_TYPE_AUTO`

Makes a call using the modem type and then automatically retries the call using the voice type if the other end cannot accept modem calls.

call_mode

Specifies a Boolean value that indicates whether the outbound call expects to receive a fax transmission when the connection completes. Set the field as follows:

`FALSE`	Indicates that the outbound call does not expect to receive a fax.
`TRUE`	Indicates that the outbound call expects to receive a fax after establishing the connection.

phonenum

Specifies a pointer to a null-terminated ASCII string that identifies the phone number to expect or dial. Dial strings (phone numbers) may be up to a maximum of 255 characters (`ECC_MAX_DIGIT_STR` - 1).

An application can use the *phonenum* field as input to the *BfvCallRingDetect* function to implement basic inbound call routing for an IP-enabled application. Unlike applications for PSTN line types, any channel in the system can answer incoming IP calls. To route an inbound IP-enabled (SIP and H.323) call, use this field to pass in a DID phone number that the incoming call must match before the system presents the call to the application (see *BfvCallRingDetect on page 297*).

PSTN Telephony

The dial string field supports the following digits and control characters. Invalid characters are ignored; upper and lower case letters are equivalent. Some protocols ignore control characters and only accept DTMF characters.

Valid Digits and Control Characters

0 - 9	Dials digits ‘0’ through ‘9’.
# (pound)	Dials a pound.
* (asterisk)	Dials an asterisk.
A - D	Sends the DTMF tone corresponding to the specified alphabetic character.
p	Changes the current or default dialing mode from tone dialing to pulse dialing.
t	Changes the current or default dialing mode from pulse dialing to tone dialing.
w	Waits for dial tone.
, (comma)	Causes a 1-second pause.
; (semicolon), i or I	Causes a 5-second pause. To create longer pauses, string any of these characters together.
! (exclamation point)	Sends a hook flash on analog and T1 robbed bit modules.

Note: In an analog environment or when using a T1 robbed bit FXS loopstart or E1 CAS loopstart protocol, the ‘w’ character means wait for dial tone. All other protocols ignore the ‘w’ and ‘i’ characters. Only analog environments and T1 robbed bit or E1 CAS protocols use the ‘p’, ‘t’, comma and semicolon characters.

IP Telephony

For IP outbound calls using the H.323 protocol:

- `Phone#@IP Address:Port#`

  If the receiving side does not require a phone number, a value for `Phone#` is optional. Also, `:Port#` is optional — the Bfv API uses 1720 as the default port value.

- `TA:IP Address:Port#,Phone#`

  `:Port#` is optional — the Bfv API uses 1720 as the default port value. If the receiving side does not require a phone number, a value for `Phone#` is optional.

- `Name:<name of person to dial>`

  Use this form only if using gatekeeper

- `E164alias:7894561234`

  Use this form only if using gatekeeper

Note: DNS lookups are not supported in H.323. You can use an H.323 or E.164 alias in conjunction with a gatekeeper to provide similar functionality.

Examples

```
4082345555@10.155.89.6:175
4082345555@10.155.89.6

TA:10.155.89.6:175,4082345555
TA:10.155.89.6,4082345555
TA:10.155.89.6

Name:Fred Smith
E164alias:4082345555
```

For IP outbound calls using the SIP protocol:

```
Phone#@IP Address:Port#
```

If the receiving side does not require a phone number, a value for Phone# is optional. Also, :Port# is optional - the Bfv API uses 5060 as the default port value.

Examples (IPv4 Addresses)

```
4082345555@10.155.89.7:175
4082345555@10.155.89.7
```

Examples (IPv6 Addresses)

```
4082345555@[2000::2ef3:1dff:ea3]:175
4082345555@[fe80::1f4:189c:74da:69f7]
```

Note: IPv6 addresses must be enclosed in brackets. In addition, if a link-local IPv6 address is specified, the Scope ID should be omitted from the address.

DTMF Post Dialing

For all types of IP calls, the character '&' (ampersand) may be included to initiate post-dialing. This character indicates that the rest of phonenum specifies a sequence of DTMF digits to be "post-dialed" after messages from the remote side indicate the call is proceeding towards connecting.

Within the post-dial string, all dialing characters listed for PSTN Telephony are allowed except for 'p', 't', 'w', and '!'. The appearance of an additional '&' will terminate processing of the string.

Post-dialing of the specified digits will occur upon the first receipt of one of the following IP call control messages:

- SIP -- 183 Progress
- SIP -- 200 OK
- H.323 -- Progress
- H.323 -- Connect

The post-dial feature is controlled by the user configuration file parameter *post_dialing_enable*. If the feature is disabled by that parameter, then the ampersand has no special effect and the entire phonenum field is used as is.

Note: If a program accepts a phone number on the command line, the phone number will likely need to be quoted if it contains an ampersand, since this is a special character on most OSes. (Use double quotes (") on Windows, or single (') or double (") quotes on Linux.)

state

Returns the current state of the call. Values are:

`BST_ALERTING`

Detected an incoming call.

`BST_CALL_DETECTED`

Detected an incoming call but the call is not ready to be answered.

`BST_CLEAR_CALL`

Attempting to clear an active call on the channel.

`BST_CONNECTED`

Call is connected.

`BST_DIAL`

Attempted an outgoing call.

`BST_DIAL_COMPLETE`

Waiting for the remote end to answer the call.

`BST_DOWN`

Phone line or trunk is out of order.

`BST_IDLE`

Channel does not have an active call.

`BST_WAIT_FOR_CALL`

Waiting to receive a call.

res

Returns a result (`RES`) structure to the caller that contains status information about the call. See *Result Structures in Volume 6, Appendix B* for return values.

cres

Returns a `CALL_RES` structure that contains specific information about the call, particularly calls made over ISDN lines. See *Appendix B, CALL_RES Structure Parameters* in *Volume 6*, for return values.

func

Specifies a pointer reference to a user-supplied integer function to call during call progress. Call this user-defined function in a loop until the function:

1. Returns a final call progress result or
2. Indicates the end of call progress by returning a value of 1.

arg

Provides an input argument for the *func* feature field. This field accepts a `NULL` setting if the user-defined function does not need an argument.

btcall_file

Specifies a null-terminated string that identifies the full path and file name to the *btcall.cfg* user-defined configuration file. The field allows a maximum of 256 characters (`MAX_PATH`).

set_log_file

Specifies a Boolean value that defines the validity of the *log_file* field as follows:

`FALSE`	Ignores the *log_file* field.
`TRUE`	Signifies that the *log_file* field contains a valid file name for the log file.

log_file

Specifies a null-terminated string that identifies the full path and file name of the log file. The field allows a maximum of 256 characters (`MAX_PATH`). Setting a value in this field overrides the log file named in the call control configuration file.

lp_second_channel

Specifies a reference to the *lp* of the second channel for the transfer process to use when the protocol requires two B-channels to transfer a call. For example, when a protocol uses the Release Link Trunk (RLT) feature that requires two B-channels to process a call transfer, the application must specify this field to make a call transfer.

Note: The SR140 does not support this field.

enquiry_call

Specifies a Boolean value that indicates whether the function sets up the outbound call to handle it as a transfer from the called party.

Note: The SR140 does not support this field.

Set the field as follows:

FALSE	Indicates that the outbound call is not set up to make a transfer.
TRUE	Indicates that the outbound call expects to make a transfer from the called party.

When the application sets the value in this field to TRUE and the line only supports single B-channel call transfer, the application must put the line on hold first using the *BfvCallHold* function. The application does not need to use the *BfvCallHold* function when the line support provides two B-channels for the transfer.

calls_on_hold

Returns the number of calls that the channel has on hold.

Note: The SR140 does not support this field.

args.ie_count

Specifies the number of custom information elements (IE) to send.

args.ie_length

Specifies the number of bytes that the custom *args.ie_data* field contains.

args.ie_data

Specifies an array of hexadecimal characters that indicate the content of the custom IE.

cause_location

Returns an ISDN-defined code that indicates the originator (local or remote) of the failure notification or call teardown (see *cause on page 441*).

name_ident

Specifies text identifying the name of the calling party if provided by the network. The field allows a maximum of 50 characters (ECC_MAX_NAME_STR).

name_char_set

Specifies the international standard specification (ISOxxx) of the character set in use. Values are:

NAME_CHAR_SET_UNKNOWN-1

Unknown character set in use.

NAME_CHAR_SET_NOT_INCLUDED0

Name does not identify a character set and the API does not send one.

NAME_CHAR_SET_ISO8859_11

Specifies use of character set defined by ISO 8859-1 international standard.

`NAME_CHAR_SET_ISO8859_23`

Specifies use of character set defined by ISO 8859-2 international standard.

`NAME_CHAR_SET_ISO8859_34`

Specifies use of character set defined by ISO 8859-3 international standard.

`NAME_CHAR_SET_ISO8859_45`

Specifies use of character set defined by ISO 8859-4 international standard.

`NAME_CHAR_SET_ISO8859_56`

Specifies use of character set defined by ISO 8859-5 international standard.

`NAME_CHAR_SET_ISO8859_77`

Specifies use of character set defined by ISO 8859-7 international standard.

`NAME_CHAR_SET_ISO10646_BMP8`

Specifies use of character set defined by ISO 10646-1 and ITU-T Recommendation X.680 international standards.

`NAME_CHAR_SET_ISO10646_UTF9`

Specifies use of character set defined by UTF-8-STRING Annex R in ISO 10646-1 international standard.

disable_auto_sw_connect

Specifies a Boolean value that indicates whether to turn the automatic switch connection feature on or off when completing a two-channel call transfer that requires connecting the two B channels together (QSIG protocol). In the *BfvCallWaitTransferComplete* function, set this field as follows:

`FALSE`	Automatically makes the switch connection for a call transfer that requires two B-channels connected together.
`TRUE`	Turns off the automatic switch connection capability for the two-channel call transfer. The application must use the *BfvCallSwConnect* function to connect the two B channels together to make a call transfer.

connected_num

Specifies a structure of type CONNECTED_NUM, containing information about the connected number that the function sends or receives from the network as part of the CONNECT message. The structure contains the following fields:

```
typedef struct {
     char addr [MAX_CONN_NUM];
     unsigned char connected_num_type;
     unsigned char connected_num_plan;
     unsigned char connected_num_presentation;
     unsigned char connected_num_screening;
} CONNECTED_NUM
```

The fields are defined as follows:

addr

Specifies a null-terminated string of up to 31 characters (MAX_CONN_NUM) that provides the telephone number of the connected party.

connected_num_type

Specifies a valid value for the type of telephone number in use depending on the value selected for *connected_num_plan*. Values are:

ECC_NUM_TYPE_ABBREVIATED
: Indicates that the port uses an abbreviated numbering type.

ECC_NUM_TYPE_INTERNATIONAL
: Indicates that the port uses an international numbering type.

ECC_NUM_TYPE_NATIONAL
: Indicates that the port uses a national numbering type.

ECC_NUM_TYPE_SUBSCRIBER
: Indicates that the port uses a subscriber numbering type.

ECC_NUM_TYPE_UNKNOWN
: Indicates that the port uses an unknown numbering type.

connected_num_plan

Specifies a valid value for the type of numbering plan in use. Values are:

ECC_NUM_PLAN_UNKNOWN
: Indicates that the port uses an unknown numbering plan.

ECC_NUM_PLAN_ISDN
: Indicates that the port uses an ISDN numbering plan.

ECC_NUM_PLAN_TELEPHONY
: Indicates that the port uses a telephony numbering plan.

ECC_NUM_PLAN_PRIVATE

Indicates that the port uses a private numbering plan.

connected_num_presentation

See *calling_party_ presentation on page 442* for the description and values for this field.

connected_num_screening

See *calling_party_screening on page 442* for the description and values for this field.

override_calling_plan

Specifies the calling party telephone numbering plan used for outbound calls. Values are:

ECC_NUM_PLAN_UNKNOWN

Indicates that the port uses an unknown numbering plan.

ECC_NUM_PLAN_ISDN

Indicates that the port uses an ISDN numbering plan.

ECC_NUM_PLAN_TELEPHONY

Indicates that the port uses a telephony numbering plan.

ECC_NUM_PLAN_PRIVATE

Indicates that the port uses a private numbering plan.

The value specified will only be active if it is ORed with the symbol *OVERRIDE_NUMBERING_FLAG*.

override_calling_type

Specifies the type of calling party telephone number used for outbound calls. Values are:

ECC_NUM_TYPE_UNKNOWN

Indicates that the port uses an unknown numbering type.

ECC_NUM_TYPE_INTERNATIONAL

Indicates that the port uses an international numbering type.

ECC_NUM_TYPE_NATIONAL

Indicates that the port uses a national (North American) numbering type.

ECC_NUM_TYPE_SUBSCRIBER

Indicates that the port uses a subscriber numbering type.

ECC_NUM_TYPE_ABBREVIATED

Indicates that the port uses an abbreviated numbering type.

The value specified will only be active if it is ORed with the symbol *OVERRIDE_NUMBERING_FLAG*.

override_called_plan

Specifies the called party telephone numbering plan used for outbound calls. Values are:

ECC_NUM_PLAN_UNKNOWN

Indicates that the port uses an unknown numbering plan.

ECC_NUM_PLAN_ISDN

Indicates that the port uses an ISDN numbering plan.

ECC_NUM_PLAN_TELEPHONY

Indicates that the port uses a telephony numbering plan.

ECC_NUM_PLAN_PRIVATE

Indicates that the port uses a private numbering plan.

The value specified will only be active if it is ORed with the symbol *OVERRIDE_NUMBERING_FLAG*.

override_called_type

Specifies the type of called party telephone number used for outbound calls. Values are:

ECC_NUM_TYPE_UNKNOWN

Indicates that the port uses an unknown numbering type.

ECC_NUM_TYPE_INTERNATIONAL

Indicates that the port uses an international numbering type.

ECC_NUM_TYPE_NATIONAL

Indicates that the port uses a national (North American) numbering type.

ECC_NUM_TYPE_SUBSCRIBER

Indicates that the port uses a subscriber numbering type.

ECC_NUM_TYPE_ABBREVIATED

Indicates that the port uses an abbreviated numbering type.

The value specified will only be active if it is ORed with the symbol *OVERRIDE_NUMBERING_FLAG*.

subcause

Specifies or returns a SIP response code that provides the reason why the call failed or terminated. The definitions of these response codes are specified in RFC 3261.

call_transport

Specifies or returns a value for SIP calls only that indicates the transport protocol from one of the following:

TRANSPORT_TYPE_UDP

Indicates User Datagram Protocol (UDP) as the transport protocol for the SIP call.

TRANSPORT_TYPE_TCP

Indicates Transmission Control Protocol (TCP) as the transport protocol for the SIP call. In order to use this setting for outbound SIP calls, TCP protocol support must be enabled in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

TRANSPORT_TYPE_DEFAULT

Indicates the default call transport, either UDP or TCP to use for an outbound SIP call. If the default call transport is not explicitly specified in the call control configuration file, UDP will be used. Otherwise, the transport protocol used for the outbound SIP call will be what's specified in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

num_user_sip_headers

Specifies the number of entries in the array of *BT_USER_SIP_HEADER* structures referenced by the *user_sip_headers* field.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

user_sip_headers

Specifies a reference to an array of *BT_USER_SIP_HEADER* structures that specify non-standard SIP header names and values to add to the initial SIP INVITE of an outbound call. The number of entries in the array is specified by the *num_user_sip_headers* field. If the value specified by *num_user_sip_headers* is 0, this field is ignored.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

fax_media_feature_tag

Specify "sip.fax" media feature tag value to add to Accept-Contact header in outbound SIP INVITE request. Use one of the following values for this field:

BT_FAX_MEDIA_FEATURE_TAG_DEFAULT

Set "sip.fax" media feature tag to a default value based on the *fax_transport_protocol* parameter value in the *t38parameters* section of the *callctrl.cfg* file as specified in the following table:

fax_transport_protocol value	"sip.fax" value
t38_never	passthrough
t38_only	t38
t38_first	t38
Not specified in *callctrl.cfg* file	t38

BT_FAX_MEDIA_FEATURE_TAG_T38

Set" sip.fax" media feature tag to "t38".

BT_FAX_MEDIA_FEATURE_TAG_PASSTHROUGH

Set "sip.fax" media feature tag to "passthrough".

BT_FAX_MEDIA_FEATURE_TAG_DISABLED

Do not add "sip.fax" media feature tag to transmitted SIP INVITE message.

In order to use this field for outbound SIP calls, RFC 6913 feature support must be enabled in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

fallback_rtp_reinvite

Specifies whether or not a SIP RTP reINVITE should be transmitted for G.711 fallback mode if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable). Valid values are:

BT_FALLBACK_RTP_REINVITE_DEFAULT

Default to the setting specified in the call control configuration file.

BT_FALLBACK_RTP_REINVITE_DISABLE

Do not transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

BT_FALLBACK_RTP_REINVITE_ENABLE

Transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

When the application sets the value in this field to *BT_FALLBACK_RTP_REINVITE_DEFAULT*, the functionality will default to the setting specified in the call control configuration file (*callctrl.cfg*) by the *g711_fallback_rtp_reinvite* parameter. If the *g711_fallback_rtp_reinvite* parameter isn't specified in the call control configuration file, then the default functionality is *BT_FALLBACK_RTP_REINVITE_DISABLE*.

An application can override the value specified in the call control configuration file by the *g711_fallback_rtp_reinvite* parameter by setting this field to a value of either *BT_FALLBACK_RTP_REINVITE_DISABLE* or *BT_FALLBACK_RTP_REINVITE_ENABLE*.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_DISABLE* will prevent transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_ENABLE* will result in transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable) and the fax transport protocol (*fax_transport_protocol*) parameter specified in the call control configuration file is set to *t38_first*. The SDP settings in the SIP RTP reINVITE will be the same RTP codec settings initially used to establish the call.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol or PSTN line types.

sip_header_list_len

Specifies the size of the memory buffer pointed to by the *sip_header_list* field. This field should be set when the Bfv application wants to retrieve the SIP header information specified in an inbound SIP INVITE request or 183 Session Progress response used to establish a SIP call.

On input to the *BfvCallWaitForSetup* or *BfvCallWaitForComplete* functions, this field should be set to the size of the memory buffer pointed to by the *sip_header_list* field.

Upon return from the *BfvCallWaitForSetup* or *BfvCallWaitForComplete* function, if a SIP call or 183 Session Progress message has successfully been received, this field will be set to the amount of data in the *sip_header_list* memory buffer that has been populated with SIP header data. If the memory buffer pointed to by *sip_header_list* is not large enough to hold all the SIP header data from the inbound SIP INVITE request or 183 Session Progress response, then *sip_header_list_len* will be set to a value of *SIP_HEADER_INVALID_LEN*. In this case, the memory buffer pointed to by *sip_header_list* will only be populated with complete SIP header name/header value pairs that fit in the buffer.

For example, if there are 10 SIP headers in the inbound SIP INVITE request or 183 Session Progress response, but the memory buffer pointed to by *sip_header_list* can only hold enough data for 9 complete SIP header name/header value pairs retrieved from the SIP INVITE request or 183 Session Progress with 30 bytes of the buffer unused, then the *sip_header_list_len* field will be set to a value of *SIP_HEADER_INVALID_LEN* and only 9 complete SIP header name/header value pairs will be returned in the *sip_header_list* buffer. No partial or incomplete SIP header information from the 10th header will be used to populate the remaining 30 bytes in the *sip_header_list* buffer.

If the Bfv application does not want to retrieve the SIP header information specified in an inbound SIP INVITE request or 183 Session Progress response, *sip_header_list_len* should be set to 0 and *sip_header_list* should be set to NULL prior to calling *BfvCallWaitForSetup* or *BfvCallWaitForComplete*.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

sip_header_list

Specifies a pointer to a memory buffer allocated by the Bfv application to receive SIP header data from an inbound SIP INVITE request or 183 Session Progress response used to establish a SIP call. The size of the memory buffer pointed to by *sip_header_list* should be specified in the *sip_header_list_len* field.

On input to the *BfvCallWaitForSetup* or *BfvCallWaitForComplete* function, this field should be set to the address of a memory buffer allocated by the Bfv application that will receive the SIP header data. As *sip_header_list* is a pointer to a *BT_SIP_HEADER_LIST* structure, the Bfv application memory buffer should be allocated and initialized in a manner similar to the following:

```
args_cc.sip_header_list =
      (BT_SIP_HEADER_LIST *)malloc(1000);
memset(args_cc.sip_header_list, 0, 1000);
args_cc.sip_header_list_len = 1000;
```

Upon return from the *BfvCallWaitForSetup* or *BfvCallWaitForComplete* functions, if a SIP call has successfully been received, the memory buffer pointed to by *sip_header_list* will be populated with a singly linked list of SIP header data stored in *BT_SIP_HEADER_NODE* structures starting with a *BT_SIP_HEADER_LIST* structure.

Some SIP headers that have multiple values may be returned as several SIP headers. For example, the following header:

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

Will be returned as five separate headers:

Header Name	*Header Value*
Allow	INVITE
Allow	ACK
Allow	OPTIONS
Allow	CANCEL
Allow	BYE

If some of the SIP header names in the inbound SIP INVITE request or 183 Session Progress response were specified in compact form, they may be returned to the Bfv application in long form (e.g., Content-Type header name specified in inbound SIP request as "c", returned to Bfv application as "Content-Type").

The maximum supported number of SIP headers that can be returned is 98.

If the Bfv application does not want to retrieve the SIP header information specified in an inbound SIP INVITE request or 183 Session Progress response, *sip_header_list_len* should be set to 0 and *sip_header_list* should be set to NULL prior to calling *BfvCallWaitForSetup* or *BfvCallWaitForComplete*.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

Note: The application must free this structure after use to release the memory.

Functions Using the args_cc Structure

The following tables provide a matrix of the low-level call control functions and the fields in the *args_cc* data structure. "Input" and "Output" define how the function requires or returns a value for the indicated field.

- *Table 17 on page 460*

  Includes *BfvCallAccept*, *BfvCallCtrlInit*, *BfvCallDisconnect*, *BfvCallDivert*

- *Table 18 on page 461*

  Includes *BfvCallHold*, *BfvCallReject*, *BfvCallRetrieve*, *BfvCallRingDetect*

- *Table 19 on page 462*

  Includes *BfvCallSendAlerting*, *BfvCallSetup*, *BfvCallStatus*, *BfvCallTransferComplete*

- *Table 20 on page 463*

  Includes *BfvCallWaitForAccept*, *BfvCallWaitForAlerting*, *BfvCallWaitForComplete*, *BfvCallWaitForDivert*

- *Table 21 on page 465*

  Includes *BfvCallWaitForHold*, *BfvCallWaitForRelease*, *BfvCallWaitForRetrieve*

- *Table 22 on page 466*

  Includes *BfvCallWaitForSetup*, *BfvCallWaitTransferComplete*

Table 17. How Functions Use args_cc Fields - Part 1

Field	BfvCallAccept	BfvCallCtrlInit	BfvCallDisconnect	BfvCallDivert
arg				
btcall_file		Input		
call_mode				
call_protocol_code				
call_type				
calling_party				
calling_party_presentation				
calling_party_screening				
calls_on_hold				
cause			Input	
cause_location				
connected_num	Input			
cres				Input
disable_auto_sw_connect				
enquiry_call				
func				
ie_count				
ie_data				
ie_length				
log_file		Input		
lp_second_channel				Input
mode				
name_char_set	Input			
name_ident	Input			
orig_called_num				
phonenum				
res	Output	Output	Output	Output
set_log_file		Input		
state				
subcause				
timeout				
call_transport				

Table 17. How Functions Use args_cc Fields - Part 1

Field	BfvCallAccept	BfvCallCtrlInit	BfvCallDisconnect	BfvCallDivert
num_user_sip_headers				
user_sip_headers				
fax_media_feature_tag				
fallback_rtp_reinvite				
sip_header_list_len				
sip_header_list				

Table 18. How Functions Use args_cc Fields - Part 2

Field	BfvCallHold	BfvCallReject	BfvCallRetrieve	BfvCallRingDetect
arg				
btcall_file				
call_mode				
call_protocol_code				
call_type				
calling_party				
calling_party_presentation				
calling_party_screening				
calls_on_hold				
cause		Input		
cause_location				
connected_num				
cres				
disable_auto_sw_connect				
enquiry_call				
func				
ie_count				
ie_data				
ie_length				
log_file				
lp_second_channel				
mode				Input
name_char_set				
name_ident				

Table 18. How Functions Use args_cc Fields - Part 2

Field	BfvCallHold	BfvCallReject	BfvCallRetrieve	BfvCallRingDetect
orig_called_num				
phonenum				Input
res	Output	Output	Output	Output
set_log_file				
state				
subcause		Input		
timeout				
call_transport				
num_user_sip_headers				
user_sip_headers				
fax_media_feature_tag				
fallback_rtp_reinvite				
sip_header_list_len				
sip_header_list				

Table 19. How Functions Use args_cc Fields - Part 3

Field	BfvCallSend Alerting	BfvCallSetup	BfvCallStatus	BfvCallTransfer Complete
arg				
btcall_file				
call_mode				
call_protocol_code		Input		
call_type		Input		
calling_party		Input		
calling_party_presentation		Input		
calling_party_screening		Input		
calls_on_hold			Output	
cause				
cause_location				
connected_num				
cres				
disable_auto_sw_connect				
enquiry_call		Input		
func				

Table 19. How Functions Use args_cc Fields - Part 3

Field	BfvCallSend Alerting	BfvCallSetup	BfvCallStatus	BfvCallTransfer Complete
ie_count		Input		
ie_data		Input		
ie_length		Input		
log_file				
lp_second_channel				Input
mode				
name_char_set		Input		
name_ident		Input		
orig_called_num				
phonenum		Input		
res	Output	Output	Output	Output
set_log_file				
state			Output	
subcause				
timeout				
call_transport		Input		
num_user_sip_headers		Input		
user_sip_headers		Input		
fax_media_feature_tag		Input		
fallback_rtp_reinvite		Input		
sip_header_list_len				
sip_header_list				

Input

Table 20. How Functions Use args_cc Fields - Part 4

Field	BfvCallWaitFor Accept	BfvCallWaitFor Alerting	BfvCallWaitFor Complete	BfvCallWaitFor Divert
arg			Input	
btcall_file				
call_mode			Input	
call_protocol_code			Input	
call_type				
calling_party				
calling_party_presentation				

Table 20. How Functions Use args_cc Fields - Part 4

Field	BfvCallWaitFor Accept	BfvCallWaitFor Alerting	BfvCallWaitFor Complete	BfvCallWaitFor Divert
calling_party_screening				
calls_on_hold				
cause	Output	Output	Output	
cause_location	Output	Output	Output	
connected_num				
cres			Output	
disable_auto_sw_connect				
enquiry_call				
func			Input	
ie_count				
ie_data				
ie_length				
log_file				
lp_second_channel				
mode				
name_char_set				
name_ident				
orig_called_num				
phonenum				
res	Output	Output	Output	Output
set_log_file				
state				
subcause	Output	Output	Output	
timeout	Input	Input	Input	Input
call_transport			Output	
num_user_sip_headers				
user_sip_headers				
fax_media_feature_tag				
fallback_rtp_reinvite				
sip_header_list_len			Input / Output	
sip_header_list			Input / Output	

Table 21. How Functions Use args_cc Fields - Part 5

Field	BfvCallWaitForHold	BfvCallWaitFor Release	BfvCallWaitFor Retrieve
arg			
btcall_file			
call_mode			
call_protocol_code			
call_type			
calling_party			
calling_party_presentation			
calling_party_screening			
calls_on_hold			
cause		Output	
cause_location		Output	
connected_num			
cres			
disable_auto_sw_connect			
enquiry_call			
func			
ie_count			
ie_data			
ie_length			
log_file			
lp_second_channel			
mode			
name_char_set			
name_ident			
orig_called_num			
phonenum			
res	Output	Output	Output
set_log_file			
state			
subcause		Output	
timeout		Input	
call_transport			

Table 21. How Functions Use args_cc Fields - Part 5

Field	BfvCallWaitForHold	BfvCallWaitFor Release	BfvCallWaitFor Retrieve
num_user_sip_headers			
user_sip_headers			
fax_media_feature_tag			
fallback_rtp_reinvite			
sip_header_list_len			
sip_header_list			

Table 22. How Functions Use args_cc Fields - Part 6

Field	BfvCallWaitForSetup	BfvCallWaitTransferComplete
arg		
btcall_file		
call_mode		
call_protocol_code		
call_type		
calling_party		
calling_party_presentation	Output	
calling_party_screening	Output	
calls_on_hold		
cause		
cause_location		
connected_num		
cres	Output	
disable_auto_sw_connect		Input
enquiry_call		
func		
ie_count		
ie_data		
ie_length		
log_file		
lp_second_channel		
mode		
name_char_set		
name_ident		

Table 22. How Functions Use args_cc Fields - Part 6

Field	BfvCallWaitForSetup	BfvCallWaitTransferComplete
orig_called_num	Output	
phonenum		
res	Output	Output
set_log_file		
state		
subcause		
timeout	Input	
call_transport	Output	
num_user_sip_headers		
user_sip_headers		
fax_media_feature_tag		
fallback_rtp_reinvite		
sip_header_list_len	Input/Output	
sip_header_list	Input/Output	

High-Level Call Control (args_telephone)

The high-level call control functions use fields of the *args_telephone* data structure to provide call control parameters for the called function. The structure contains the following fields:

```
struct args_telephone
{
    RES res;
    CALL_RES call_res;
    MCONST char *phonenum;
    int call_protocol_code;
    long timeout;
    int  call_mode;
    M_CB1 int (* M_CB2 func)(MILL_LINE *lp, char *arg);
    char *arg;
    unsigned char type;
    unsigned char data;
    MCONST char *db_file;
    MCONST RES *db_res;
    long wait_time;
    int reason;
    M_CB1 int (* M_CB2 db_func)(struct dialdb *dbentry, char *arg);
    char *raw_number;
    int orig_answer;
    unsigned report_cadence;
    unsigned report_freq;
    unsigned req_report_time;
    unsigned template_number;
    unsigned prog_freq;
    unsigned prog_cadence;
    unsigned diss_mode;
    unsigned short frequencies[2];
    unsigned duration;
    int level;
    unsigned short cadence_pattern[4];
    struct {
      unsigned short freq;
      short level;
    } freq_data[3];
    int diss_only;
    unsigned cause_code;
    MILL_LINE *async_lp;
    int transfer_mode;
    int transfer_group;
    MILL_LINE *lp_second_channel;
    int supervised;
    int transfer_line_state;
    int hold_call;
    unsigned subcause;
    unsigned cause_location;
    enum TRxTransportType call_transport;
    unsigned num_user_sip_headers;
    struct BT_USER_SIP_HEADER *user_sip_headers;
    int fax_media_feature_tag;
    unsigned fallback_rtp_reinvite;
    unsigned sip_header_list_len;
    BT_SIP_HEADER_LIST *sip_header_list;
};
```

Field	Description
res	Returns a result (`RES`) structure to the caller that contains status information about the call. See *Result Structures* in *Volume 6, Appendix B* for return values.
call_res	Returns a `CALL_RES` structure that contains specific information about the call, particularly calls made over ISDN lines. See *CALL_RES Structure Parameters* in *Volume 6, Appendix B* for return values.
phonenum	Specifies a pointer to a null-terminated ASCII string that identifies the phone number to expect or dial. Dial strings (phone numbers) may be up to a maximum of 255 characters (`ECC_MAX_DIGIT_STR` - 1). An application can use the *phonenum* field as input to the *BfvLineWaitForCall* function to implement basic inbound call routing for an IP-enabled application. Unlike applications for PSTN line types, any channel in the system can answer incoming IP calls. To route an inbound IP-enabled (SIP or H.323) call, use this field to pass in a DID phone number that the incoming call must match before the system presents the call to the application (see *BfvLineWaitForCall on page 408*). *PSTN Telephony* The dial string field supports the following digits and control characters. Invalid characters are ignored; upper and lower case letters are equivalent. Some protocols ignore control characters and only accept DTMF characters.

Valid Digits and Control Characters

0 - 9	Dials digits ‘0’ through ‘9’.
# (pound)	Dials a pound.
* (asterisk)	Dials an asterisk.
A - D	Sends the DTMF tone corresponding to the specified alphabetic character.
p	Changes the current or default dialing mode from tone dialing to pulse dialing.
t	Changes the current or default dialing mode from pulse dialing to tone dialing.
w	Waits for dial tone.
, (comma)	Causes a 1-second pause.
; (semicolon), i or I	Causes a 5-second pause. To create longer pauses, string any of these characters together.
! (exclamation point)	Sends a hook flash on analog and T1 robbed bit modules.

Note: In an analog environment or when using a T1 robbed bit FXS loopstart or E1 CAS loopstart protocol, the ‘w’ character means wait for dial tone. All other protocols ignore the ‘w’ and ‘i’ characters. Only analog environments and T1 robbed bit or E1 CAS protocols use the ‘p’, ‘t’, comma and semicolon characters.

IP Telephony

For IP outbound calls using the H.323 protocol:

- `Phone#@IP Address:Port#`

  If the receiving side does not require a phone number, a value for `Phone#` is optional. Also, `:Port#` is optional — the Bfv API uses 1720 as the default port value.

- `TA:IP Address:Port#,Phone#`

  `:Port#` is optional — the Bfv API uses 1720 as the default port value. If the receiving side does not require a phone number, a value for `Phone#` is optional.

- `Name:<name of person to dial>`

  Use this form only if using gatekeeper

- `E164alias:7894561234`

  Use this form only if using gatekeeper

Note: DNS lookups are not supported in H.323. You can use an H.323 or E.164 alias in conjunction with a gatekeeper to provide similar functionality.

Examples

```
4082345555@10.155.89.6:175
4082345555@10.155.89.6

TA:10.155.89.6:175,4082345555
TA:10.155.89.6,4082345555
TA:10.155.89.6

Name:Fred Smith
E164alias:4082345555
```

For IP outbound calls using the SIP protocol:

```
Phone#@IP Address:Port#
```

If the receiving side does not require a phone number, a value for Phone# is optional. Also, :Port# is optional - the Bfv API uses 5060 as the default port value.

Examples (IPv4 Addresses)

```
4082345555@10.155.89.7:175
4082345555@10.155.89.7
```

Examples (IPv6 Addresses)

```
4082345555@[2000::2ef3:1dff:ea3]:175
4082345555@[fe80::1f4:189c:74da:69f7]
```

Note: IPv6 addresses must be enclosed in brackets. In addition, if a link-local IPv6 address is specified, the Scope ID should be omitted from the address.

DTMF Post Dialing

For all types of IP calls, the character '&' (ampersand) may be included to initiate post-dialing. This character indicates that the rest of phonenum specifies a sequence of DTMF digits to be "post-dialed" after messages from the remote side indicate the call is proceeding towards connecting.

Within the post-dial string, all dialing characters listed for PSTN Telephony are allowed except for 'p', 't', 'w', and '!'. The appearance of an additional '&' will terminate processing of the string.

Post-dialing of the specified digits will occur upon the first receipt of one of the following IP call control messages:

- SIP -- 183 Progress
- SIP -- 200 OK
- H.323 -- Progress
- H.323 -- Connect

The post-dial feature is controlled by the user configuration file parameter *post_dialing_enable*. If the feature is disabled by that parameter, then the ampersand has no special effect and the entire phonenum field is used as is.

Note: If a program accepts a phone number on the command line, the phone number will likely need to be quoted if it contains an ampersand, since this is a special character on most OSes. (Use double quotes (") on Windows, or single (') or double (") quotes on Linux.)

call_protocol_code

Specifies a calling protocol for the module that determines how and when the module reports the results of call progress analysis. Set the value of this field to:

`CALL_PROTOCOL_FAX`

Selects the fax protocol. This setting requests the module to report the results from the module's call progress analysis along with the raw data. The module reports the results as soon as it establishes the fax connection or encounters a busy condition.

`CALL_PROTOCOL_RAW`

Selects a value that requests the module to report the raw high and low call progress results without performing any analysis.

`CALL_PROTOCOL_VOICE`

Selects the voice protocol. This setting requests the module to report the results from the module's call progress analysis along with the raw data. The module reports the results as soon as it detects a human or other answer condition.

`CALL_PROTOCOL_FAX_NO_RAW`

`CALL_PROTOCOL_VOICE_NO_RAW`

Selects either the fax or voice protocol, requesting the module to report the results from the module's call progress analysis without including the raw data.

timeout

Specifies a value that determines the length of time to wait for a call control function to complete. Values are:

`0` Indicates waiting forever with no timeout.

`nonzero` Indicates the number of milliseconds to wait for the function to complete.

call_mode

Specifies a Boolean value that indicates whether the outbound call expects to receive a fax transmission when the connection completes. Set the field as follows:

`FALSE` Indicates that the outbound call does not expect to receive a fax.

`TRUE` Indicates that the outbound call expects to receive a fax after establishing the connection.

func

Specifies a pointer to a user-supplied integer function to call during call progress. Call this user-defined function in a loop until the function:

1. Returns a final call progress result or
2. Indicates the end of call progress by returning a value of 1.

arg

Provides an input argument for the *func* or *db_func* feature field. This field accepts a `NULL` setting if the user-defined function does not need an argument.

type

Contains one of the following values that indicates the call progress type:

`CP_TYPE_LOW`

Low call progress. The data indicates the length of time in 5-msec units that the output of the call progress filter was low.

`CP_TYPE_HIGH`

High call progress. The data indicates the length of time in 5-msec units that the output of the call progress filter was high.

`CP_TYPE_BOARD_DET`

A channel-determined call progress result. See *Call Progress Notes in Volume 6, Appendix F*, for interpretation of the data values.

`CP_TYPE_CED`

Detection of a fax answer tone. The *args.data* field does not apply.

`CP_TYPE_FREQUENCY`

Frequency information. The *args.freq_data* field contains the frequency information. Frequency information is only used when frequency reporting is enabled in the *BfvLineCallProgressEnable* function.

`CP_TYPE_CADENCE`

Cadence information. The *args.cadence_pattern* field contains the cadence information. Cadence information is only used when cadence reporting is enabled in the *BfvLineCallProgressEnable* function.

data

Contains the associated data that corresponds to the call progress type.

db_file

Specifies the name of the dialing database file.

Many countries have dialing restrictions that require applications to maintain a database of information about previously called fax telephone numbers. If the application dials a fax machine in one of these countries, it must use the dialing database functions (see *Chapter , Dialing Database Functions,* on *page 420* and *Volume 6, Appendix G, Country-Specific Parameter Files*) to avoid violating the target country's PTT regulations.

db_res

Specifies a pointer to a `RES` structure that contains the result of the last attempt to call the number specified in the *args.raw_number* field.

wait_time

Returns a value indicating the amount of time in seconds that the function waited before dialing.

reason

Returns a value indicating why the system delayed dialing for the period indicated in *args.wait_time*.

This field returns one of the following values:

DL_REASON_NONE

DL_REASON_WRONG_DELAY

DL_REASON_UNSUCC_DELAY

DL_REASON_BLACK_PRE

DL_REASON_BLACK_MANUAL

DL_REASON_BLACK_UNSUCC

DL_REASON_BLACK_SUM

DL_REASON_UNSUCC_LMT

DL_REASON_SUM_LMT

DL_REASON_TRK_UNS_DELAY

DL_REASON_UNKNOWN

db_func

Specifies a pointer to a user-supplied integer function that the application must call once for each entry in the dialing database.

The value of *args.db_func* cannot be NULL. Call the function as:

`(*args.db_func) (dbentry, args.arg)` where `dbentry` is a *struct dialdb* * pointer to a structure that describes a database entry, and `arg` is the user-supplied argument.

raw_number

Specifies a string that contains the telephone number to dial. This string must contain only the actual telephone number and omit long distance, trunk, or other codes. Valid characters include:

0 – 9

(pound or number symbol)

* (star or asterisk symbol)

The function ignores any other character.

orig_answer

Specifies one of the following values that indicates the purpose of taking the line off hook and making a connection. The function ignores the setting when the line operates in ISDN mode.

BT_ORIGINATE

Sets the line state to CONNECTED to make an outbound call.

BT_ANSWER

Sets the line state to CONNECTED to answer an incoming call.

report_cadence

Specifies a Boolean value that enables or disables cadence analysis and reporting. This mode enables full call progress mode only, not limited call progress mode (DISS).

Specify a `TRUE` value for this field only for testing and troubleshooting purposes. When you enable this field, you disable most other call progress analysis.

`FALSE`	Disables cadence analysis and reporting for the outbound call.
`TRUE`	Enables cadence analysis and reporting for the outbound call. Use this value only for troubleshooting or testing purposes.

report_freq

Specifies a Boolean value that enables or disables frequency analysis and reporting. This mode enables full call progress mode only, not limited call progress mode (DISS).

Specify a `TRUE` value for this field only for testing and troubleshooting purposes. When you enable this field, you disable most other call progress analysis.

`FALSE`	Disables frequency analysis and reporting for the outbound call.
`TRUE`	Enables frequency analysis and reporting for the outbound call. Use this value only for troubleshooting or testing purposes.

freq_report_time

Specifies a frequency reporting interval in milliseconds. The typical value is 1000 (milliseconds).

template_number

Specifies the template number to program. In full call progress mode, the range of values can be 0 through 7. In limited call progress mode (DISS), set the template number to 0.

prog_freq

Specifies a Boolean value that indicates whether to use frequency programming. The application must enable this field or the *args.prog_cadence* field for programmable call progress monitoring. When enabled for frequency programming, use the *args.frequencies*, *args.duration*, and *args.level* fields to program a frequency detector. Set the field as follows:

`FALSE`	Disables frequency programming.
`TRUE`	Enables frequency programming.

prog_cadence

Specifies a Boolean value that indicates whether to use cadence analysis programming. The application must enable this field or the *args.prog_frequency* field for programmable call progress monitoring. When enabled for cadence analysis programming in full progress mode, use the *args.cadence_pattern field* to program a cadence detector. In limited call progress mode, also set values in the *args.frequencies* and *args.level* fields to program a cadence detector. Set the field as follows:

`FALSE`	Disables cadence analysis programming.
`TRUE`	Enables cadence analysis programming.

diss_mode

Specifies a Boolean value that indicates whether to use programming for the limited call progress performed during speech (DISS). Set the field as follows:

`FALSE`	Performs programming in full call progress mode.
`TRUE`	Performs programming in limited call progress mode during speech (DISS).

frequencies[2]

Specifies an array consisting of two frequency values in Hz. Use this field for programming frequency analysis in both full and limited call progress modes and for cadence analysis in limited call progress mode (DISS). The function only uses the first frequency when detecting frequencies in limited call progress mode.

duration

Specifies the minimum length of time that the system requires to recognize a tone. The unit value is milliseconds. Use this field for programming frequency analysis.

level

Specifies the minimum magnitude of the signal that the system requires to detect a tone. The unit value is cBm (centibel milliwatt). Use this field for programming frequency analysis in both full and limited call progress modes and for cadence analysis in limited call progress mode (DISS).

cadence_pattern[4]

Specifies an array of four duration values that define the cadence pattern. The unit value is milliseconds. The values represent the high/low/high/low durations that define a cadence pattern for full call progress. Use this field for programming cadence analysis when you set *args.type* with a value of `CP_TYPE_CADENCE`. In limited call progress mode (DISS), call progress only uses the first two values. For full call progress, the system only detects signals in the range of 300 – 640 Hz.

freq_data[3]

Specifies an array of three elements where each element contains tone information that consists of a frequency in Hz and a level in cBm. The three tones together describe the detected signal. Use this field when you set *args.type* with a value of `CP_TYPE_FREQUENCY`.

diss_only

Specifies a Boolean value that unconditionally enables limited call progress mode (DISS). When enabled for DISS mode, the function ignores all other call progress input options. Full duplex speech operations frequently select this call progress mode. Set this field as follows:

`FALSE`	Disables limited call progress (DISS) mode.
`TRUE`	Enables limited call progress (DISS) mode.

cause_code

Specifies or returns an ISDN-defined code that provides the reason why the call failed. See *Volume 6, Appendix D, Defining ISDN Cause Codes* for a description.

async_lp

Specifies a line pointer to the session so that the application notifies the session when the operation is complete. If the field contains a nonzero value, it enables asynchronous operation. See the developer's guide that came with your software for more information on asynchronous operation and usage.

transfer_mode

Returns a value that indicates how a line supports call transfer.

The SR140 does not support this field.

Values are:

`LINE_XFER_NONE`	Line does not support call transfer.
`LINE_XFER_SINGLE`	Line only supports single B-channel (same channel) call transfer.
`LINE_XFER_TWO_CHAN`	Line only supports two B-channel (explicit) call transfer.
`LINE_XFER_ALL`	Line supports single and two B-channel call transfers.

`LINE_XFER_TWO_CHAN_NEEDS_NAILUP` Line supports two B-channel call transfers, but the application must connect the B-channels together. Only a QSIG protocol supports this value. The API makes the connection automatically if using a high level API function, and provides the option to make the required connection automatically when you set *args.disable_auto_sw_connect* to `FALSE` for *BfvCallWaitTransferComplete*. If you set *args.disable_auto_sw_connect* to `TRUE`, the application must call the *BfvCallSwitchConnect* function to make the required B-channel connection.

transfer_group

Returns matching line values to the application that indicate which lines can be paired to perform a two B-channel call transfer. When the line values do not match, the lines cannot be paired to perform a two B-channel call transfer. Only lines that support two B-channel call transfers return valid values in this field.

The SR140 does not support this field.

lp_second_channel

A reference to the *lp* of the line to use for the enquiry call for protocols that require two B-channels to transfer a call (for example, Release Link Trunk (RLT) protocol).

The SR140 does not support this field.

Set to 0 when the transfer occurs over a single channel (for example, an analog line).

supervised

Specifies a Boolean value that determines whether the function automatically completes the transfer or returns without completing the transfer when it detects the value defined for the *args.transfer_line_state* argument.

The SR140 does not support this field.

Values are:

`FALSE` Indicates that the function automatically completes the transfer when it detects the value defined for *args.transfer_line_state*.

`TRUE` Indicates that the function returns control to the application without completing the transfer when it detects the value defined for *args.transfer_line_state*.

transfer_line_state

Specifies the call state that determines when to complete the transfer.

The SR140 does not support this field.

Values are:

`BST_DIAL_COMPLETE`

Completes the call transfer after dialing the transfer number.

`BST_ALERTING`

Completes the call transfer after detecting a ring.

`BST_CONNECTED`

Completes the call transfer when the called party answers.

hold_call

Specifies an integer value that determines whether to place the original call on hold or allow it to remain active. Use this field when transferring a call using two B channels; the field has no effect on single B-channel call transfers that must place the original call on hold to make the transfer.

The SR140 does not support this field.

Values are:

`0`	Allows the original call to remain active and does not place it on hold before making the enquiry call on the other B channel.
`nonzero`	Places the original call on hold before making the enquiry call on the other B channel.

subcause

Specifies or returns a SIP response code that provides the reason why the call failed or terminated. The definitions of these response codes are specified in RFC 3261.

cause_location

Returns an ISDN-defined code that indicates the originator (local or remote) of the failure notification or call teardown (see *cause_code*).

call_transport

Specifies or returns a value for SIP calls only that indicates the transport protocol from one of the following:

TRANSPORT_TYPE_UDP

Indicates User Datagram Protocol (UDP) as the transport protocol for the SIP call.

TRANSPORT_TYPE_TCP

Indicates Transmission Control Protocol (TCP) as the transport protocol for the SIP call. In order to use this setting for outbound SIP calls, TCP protocol support must be enabled in the call control configuration file (see *Volume 6, Appendix 6, Appendix A, Configuration Files*.)

TRANSPORT_TYPE_DEFAULT

Indicates the default call transport, either UDP or TCP to use for an outbound SIP call. If the default call transport is not explicitly specified in the call control configuration file, UDP will be used. Otherwise, the transport protocol used for the outbound SIP call will be what's specified in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

NOTE: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

num_user_sip_headers

Specifies the number of entries in the array of *BT_USER_SIP_HEADER* structures referenced by the *user_sip_headers* field.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

user_sip_headers

Specifies a reference to an array of *BT_USER_SIP_HEADER* structures that specify SIP header names and values to add to the initial SIP INVITE of an outbound call. The number of entries in the array is specified by the *num_user_sip_headers* field. If the value specified by *num_user_sip_headers* is 0, this field is ignored.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

fax_media_feature_tag

Specify "sip.fax" media feature tag value to add to Accept-Contact header in outbound SIP INVITE request. Use one of the following values for this field:

BT_FAX_MEDIA_FEATURE_TAG_DEFAULT

Set "sip.fax" media feature tag to a default value based on the *fax_transport_protocol* parameter value in the *t38parameters* section of the *callctrl.cfg* file as specified in the following table:

fax_transport_protocol value	"sip.fax" value
t38_never	passthrough
t38_only	t38
t38_first	t38
Not specified in *callctrl.cfg* file	t38

BT_FAX_MEDIA_FEATURE_TAG_T38

Set" sip.fax" media feature tag to "t38".

BT_FAX_MEDIA_FEATURE_TAG_PASSTHROUGH

Set "sip.fax" media feature tag to "passthrough".

BT_FAX_MEDIA_FEATURE_TAG_DISABLED

Do not add "sip.fax" media feature tag to transmitted SIP INVITE message.

In order to use this field for outbound SIP calls, RFC 6913 feature support must be enabled in the call control configuration file (see *Volume 6, Appendix A, Configuration Files*).

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol and PSTN line types.

fallback_rtp_reinvite

Specifies whether or not a SIP RTP reINVITE should be transmitted for G.711 fallback mode if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable). Valid values are:

BT_FALLBACK_RTP_REINVITE_DEFAULT

Default to the setting specified in the call control configuration file.

BT_FALLBACK_RTP_REINVITE_DISABLE

Do not transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

BT_FALLBACK_RTP_REINVITE_ENABLE

Transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

When the application sets the value in this field to *BT_FALLBACK_RTP_REINVITE_DEFAULT*, the functionality will default to the setting specified in the call control configuration file (*callctrl.cfg*) by the *g711_fallback_rtp_reinvite* parameter. If the *g711_fallback_rtp_reinvite* parameter isn't specified in the call control configuration file, then the default functionality is *BT_FALLBACK_RTP_REINVITE_DISABLE*.

An application can override the value specified in the call control configuration file by the *g711_fallback_rtp_reinvite* parameter by setting this field to a value of either *BT_FALLBACK_RTP_REINVITE_DISABLE* or *BT_FALLBACK_RTP_REINVITE_ENABLE*.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_DISABLE* will prevent transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_ENABLE* will result in transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable) and the fax transport protocol (*fax_transport_protocol*) parameter specified in the call control configuration file is set to *t38_first*. The SDP settings in the SIP RTP reINVITE will be the same RTP codec settings initially used to establish the call.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol or PSTN line types.

sip_header_list_len

Specifies the size of the memory buffer pointed to by the *sip_header_list* field. This field should be set when the Bfv application wants to retrieve the SIP header information specified in an inbound SIP INVITE request or 183 Session Progress response used to establish a SIP call.

On input to the *BfvLineWaitForCall* or *BfvLineOriginateCall* functions, this field should be set to the size of the memory buffer pointed to by the *sip_header_list* field.

Upon return from the *BfvLineWaitForCall* or *BfvLineOriginateCall* function, if a SIP call or 183 Session Progress message has successfully been received, this field will be set to the amount of data in the *sip_header_list* memory buffer that has been populated with SIP header data. If the memory buffer pointed to by *sip_header_list* is not large enough to hold all the SIP header data from the inbound SIP INVITE request or 183 Session Progress response, then *sip_header_list_len* will be set to a value of *SIP_HEADER_INVALID_LEN*. In this case, the memory buffer pointed to by *sip_header_list* will only be populated with complete SIP header name/header value pairs that fit in the buffer.

For example, if there are 10 SIP headers in the inbound SIP INVITE request or 183 Session Progress response, but the memory buffer pointed to by *sip_header_list* can only hold enough data for 9 complete SIP header name/header value pairs retrieved from the SIP INVITE request or 183 Session Progress with 30 bytes of the buffer unused, then the *sip_header_list_len* field will be set to a value of *SIP_HEADER_INVALID_LEN* and only 9 complete SIP header name/header value pairs will be returned in the *sip_header_list* buffer. No partial or incomplete SIP header information from the 10th header will be used to populate the remaining 30 bytes in the *sip_header_list* buffer.

If the Bfv application does not want to retrieve the SIP header information specified in an inbound SIP INVITE request or 183 Session Progress response, *sip_header_list_len* should be set to 0 and *sip_header_list* should be set to NULL prior to calling *BfvLineWaitForCall* or *BfvLineOriginateCall*.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

sip_header_list

Specifies a pointer to a memory buffer allocated by the Bfv application to receive SIP header data from an inbound SIP INVITE request or 183 Session Progress response used to establish a SIP call. The size of the memory buffer pointed to by *sip_header_list* should be specified in the *sip_header_list_len* field.

On input to the *BfvLineWaitForCall* or *BfvLineOriginateCall* functions, this field should be set to the address of a memory buffer allocated by the Bfv application that will receive the SIP header data. As *sip_header_list* is a pointer to a *BT_SIP_HEADER_LIST* structure, the Bfv application memory buffer should be allocated and initialized in a manner similar to the following:

```
args_tel.sip_header_list =
       (BT_SIP_HEADER_LIST *)malloc(1000);
memset(args_tel.sip_header_list, 0, 1000);
args_tel.sip_header_list_len = 1000;
```

Upon return from the *BfvLineWaitForCall* or *BfvLineOriginateCall* functions, if a SIP call or 183 Session Progress message has successfully been received, the memory buffer pointed to by *sip_header_list* will be populated with a singly linked list of SIP header data stored in *BT_SIP_HEADER_NODE* structures starting with a *BT_SIP_HEADER_LIST* structure.

Some SIP headers that have multiple values may be returned as several SIP headers. For example, the following header:

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

Will be returned as five separate headers:

Header Name	*Header Value*
Allow	INVITE
Allow	ACK
Allow	OPTIONS

Allow	CANCEL
Allow	BYE

If some of the SIP header names in the inbound SIP INVITE request or 183 Session Progress response were specified in compact form, they may be returned to the Bfv application in long form (e.g., Content-Type header name specified in inbound SIP request as "c", returned to Bfv application as "Content-Type").

The maximum supported number of SIP headers that can be returned is 98.

If the Bfv application does not want to retrieve the SIP header information specified in an inbound SIP INVITE request or 183 Session Progress response, *sip_header_list_len* should be set to 0 and *sip_header_list* should be set to NULL prior to calling *BfvLineWaitForCall* or *BfvLineOriginateCall*.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for all calls using the H.323 internet protocol or PSTN line types.

Note: The application must free this structure after use to release the memory.

Macros

LINE_AOC_INFO (*lp*)

This macro only applies to boards with a BRI interface or digital ports using the QSIG protocol variant.

Accesses a structure containing AOC (advice of charge) information.

Returns a structure pointer of type `AOC_INFO *`. The structure is:

```
typedef struct {
    int status;
    int charge_info;
    long charge;
} AOC_INFO;
```

After a call completes, this structure returns the AOC information. The fields have the following meanings:

status — Returns a value indicating the form of the AOC information, if available. The values are:

BT_AOC_UNAVAILABLE	0
BT_AOC_FREE	1
BT_AOC_UNITS	2
BT_AOC_CURRENCY	3

charge_info — For status `BT_AOC_UNITS`, specifies the units. The user defines the meaning of the units.

For status `BT_AOC_CURRENCY`, specifies the multiple of one of the `BT_AOC_CUR_...` values defined in a header file. The header file is *mill_api.h*.

charge — Returns the value of the charge in units or currency.

Volume 3 - Media Processing

About This Volume

- *Volume 3, Media Processing,* provides information about the following Bfv API components:
  - Signal Generation and Detection functions
  - Voice Play and Record functions
  - Infopkt file functions
  - Audio Conferencing functions

14 - Signal Generation and Detection

Not all functions in this chapter are supported on the SR140. The functions not fully supported are noted.

This chapter describes functions to generate and detect various tones, digits and call progress patterns.

With the signal generation and tone detection functions, you can:

- Play call progress signals and generate other tone groups and tone patterns.
- Get the next call progress code.
- Enable and disable `DTMF` detection.
- Discard tones from a buffer.
- Wait for a tone and return it as an `ASCII` character or return it without disturbing the buffer.
- Play a tone for a specified time.
- Play a single frequency tone.
- Replace a tone in the buffer for reuse.

Channels on Dialogic® Brooktrout® modules receive call progress signals generated by telcos and Private Branch Exchanges (PBXs) before, during, and after dialing. Then the firmware's call progress analysis process interprets them.

During call progress analysis, the firmware can report dial tone detection, ring-back, busy signals, remote fax tone detection, and other important information. Applications can use this information to determine their next course of action, to display the status of a call, or to track billing information. Applications can use postdialing results such as `HUMAN` and `BUSY` to decide what redialing strategy to use.

Channels can also generate and play `DTMF` and `MF` tone groups and single tone patterns to send to the telco or `PBX`.

The SR140 supports detection, but not generation, of RFC 2833 DTMF digits as RTP events. The SR140 will detect and process only the 16 standard telephony digits (0-9, *, #, A-D). See the *rfc_2833_enabled* parameter in *Appendix A - Configuring A Module To Use An RTP Stack*.

Note: RFC 2833 was superseded by RFC 4733. However, that update does not affect the implementation of this functionality.

Signal Generation/Detection Function Summary

Table 23 provides a high-level description for the signal generation and detection functions. Function details in alphabetical order begin on *page 492*.

Table 23. Signal Generation/Detection Function Summary

Function	Purpose	Page
BfvCPGen	Generates call progress signals.	*492*
BfvCPGenAdv	Generates call progress signals and other tone patterns.	*494*
BfvDataCP	Retrieves the next call progress code.	*498*
BfvLineCallProgressDisable	Disables call progress.	*502*
BfvLineCallProgressEnable	Enables call progress in one of three modes.	*504*
BfvLineCallProgressProgram	Programs frequency and cadence analysis parameters used during call progress monitoring.	*509*
BfvToneDetectDisable	Disables DTMF detection.	*513*
BfvToneDetectEnable	Enables DTMF detection.	*515*
BfvToneFlush	Discards all tones currently stored in the buffer.	*518*
BfvToneGet	Retrieves the next tone from the tone buffer and removes it from the buffer.	*519*
BfvTonePeek	Retrieves the next tone in the buffer without disturbing the buffer.	*521*
BfvTonePlay	Plays the tone for the specified time.	*523*
BfvTonePlayBeep	Plays a single frequency tone.	*525*
BfvToneUnget	Puts a tone at the top of the tone buffer, so it is available for the next request to retrieve *a* tone.	*528*

BfvCPGen

Purpose

Generates call progress signals.

Syntax

```
void
BfvCPGen                    (lp, args)
   BTLINE                   *lp;
   struct args_tone         *args;
```

The structure contains the following fields.

Input Field

int *cp_type;*

Output Field

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.cp_type

Indicates the type of call progress signals to generate. Valid values are:

CP_GEN_DIALTONE	0
CP_GEN_RING	1
CP_GEN_BUSY	2
CP_GEN_REORDER	20
CP_GEN_END	128

Output

Return value: None

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

To begin signal generation, call this function with a value designating a call progress signal type. To end signal generation, call the function again with the special value CP_GEN_END.

After starting, signal generation continues indefinitely until the application calls *BfvCPGen* with CP_GEN_END. The application must end signal generation before performing any other activity, for example: sending a fax, playing speech, enabling full call progress, or playing touch tones. Limited call progress (DISS) can be used.

See Also

BfvCPGenAdv

Example

```
struct args_tone args;
struct args_packet args_pkt;

BT_ZERO(args);
args.cp_type = CP_GEN_RING;
BfvCPGen(lp,&args);
BT_ZERO(args_pkt);
args_pkt.timeout = 1000L;
BfvRcvProcessPkt(lp,&args_pkt);
args.cp_type = CP_GEN_END;
BfvCPGen(lp,&args);
```

BfvCPGenAdv

Purpose

Generates and plays call progress signals, other tone groups, and tone patterns.

Syntax

```
int
BfvCPGenAdv                 (lp, args)
   BTLINE                   *lp;
   struct args_tone         *args;
```

The structure contains the following fields.

Input Fields

```
unsigned count;
struct cpgen_signal_info *signal_info;
int volume;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.count

The number of times to play the entire tone pattern.

If *args.signal_info* is NULL, the value of this argument indicates whether to wait for signal generation to finish (0) or to terminate signal generation (1).

args.signal_info

If non-NULL, this argument is a pointer to an array of cpgen_signal_info structures, each of which describes a group of tones to play.

If NULL, *BfvCPGenAdv* either completes or terminates signal generation, depending on the value in *count*.

args.volume

If nonzero, indicates that attenuation values provided represent level values in 0.1 dBm units.

Output

Return value:

0 The function executed successfully.

<0 An error occurred.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 does not support this function.

A tone consists of a frequency and an attenuation value or volume. A tone group consists of a duration, in milliseconds, and up to two tones. A tone pattern consists of one or more tone groups.

This function plays all tones in a tone group simultaneously for a specified duration. It can play several tone groups in sequence and tone patterns multiple times.

The *args.signal_info* argument is a pointer to an array of `cpgen_signal_info` structures:

```
struct cpgen_signal_info {
    unsigned play_time;
    struct cpgen_tone_info *tone_info;
};
```

Each `cpgen_signal_info` structure describes a group of tones to play simultaneously. The application must include an additional terminating `cpgen_signal_info` structure that contains a 0 in the `play_time` field.

The `play_time` field of each `cpgen_signal_info` structure contains the time, in milliseconds, to play the tones. The maximum value is 0xFFFF.

The `tone_info` field is a pointer to an array of `cpgen_tone_info` structures, each of which describes one tone:

```
struct cpgen_tone_info {
    unsigned freq;
    unsigned atten;
```

```
};
```

The application must include an additional terminating `cpgen_tone_info` structure that contains a 0 in the *freq* field.

If the array contains only the terminating structure, the function plays silence for the specified duration. The maximum number of tones permitted in a `tone_info` array (and the maximum number of tones this function can play simultaneously) is two.

The *freq* field of each `cpgen_tone_info` structure contains the frequency, in Hertz, of the tone. The valid range is 100 to 3500. The *atten* field contains the attenuation value for the tone in 0.1 dB units. The valid range is 0 to 120, where 0 represents maximum output.

To complete signal generation, call this function with a *args.signal_info* value of `NULL`. This function waits for signal generation to finish if the value of *count* is 0, or it terminates signal generation if the value of *count* is 1.

Applications can generate several common call progress signals using the following tone information:

Dial tone350 Hz and 440 Hz, continuous

Ringing440 Hz and 480 Hz, on/off

Busy 480 Hz and 620 Hz, on/off

Purity of the signal generated when this function plays the individual tones in the tone group simultaneously depends on the harmonics produced by mixing the tones' frequencies. Tones in the range of 100 to 200 Hz or 3450 to 3500 Hz might be weaker.

When an application exceeds this limit, the Bfv API returns an error indication.

The maximum number of tone groups is 20.

The application must end signal generation before performing any other activity, for example: sending a fax, playing speech, enabling full call progress, or playing touch tones. Limited call progress (`DISS`) can be used.

See Also

BfvCPGen

Example

```
/* Tone info for silence portion of ringing */
struct cpgen_tone_info silence_freq[] = {
    {0}
};

/* Tone info for ring portion of ringing */
struct cpgen_tone_info ring_freq[] = {
    {440,0},
    {480,0},
    {0}
};
/* Signal info for the entire ringing pattern */
struct cpgen_signal_info ring_info[] = {
    {1000,ring_freq},
    {3000,silence_freq},
    {0,NULL}
};

BT_ZERO(args);
args.count = 1;
args.signal_info = ring_info;
/* Play one cycle of ringing */
BfvCPGenAdv(lp, &args);
BT_ZERO(args);
/* Wait for signal generation to finish */
BfvCPGenAdv(lp, &args);
```

BfvDataCP

Purpose

Retrieves the next call progress code.

Syntax

```
int
BfvDataCP                    (lp, args)
   BTLINE                    *lp;
   struct args_telephone     *args;
```

The structure contains the following fields.

Input Fields

None

Output Fields

```
unsigned char type;
unsigned char data;
unsigned short cadence_pattern[4];
struct {
   unsigned short freq;
   short level;
   } freq_data[3];
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value:

0 New data is available and is stored in *args.type* and *args.data*.

1 No new data available.

`args.type`

Contains the call progress type. It can contain any of the following values:

`CP_TYPE_LOW` 0

Low call progress. The data indicates, in 5 ms units, the time interval in which the output of the call progress filter was low.

`CP_TYPE_HIGH` 1

High call progress. The data indicates, in 5 ms units, the time interval when the output of the call progress filter was high.

`CP_TYPE_BOARD_DET` 2

A firmware-determined call progress result.

`CP_TYPE_CED` 3

Detection of a fax answer tone. The *args.data* is not relevant.

`CP_TYPE_FREQUENCY` 4

Frequency information. The frequency information is contained in *args.freq_data*. Use this argument only when frequency reporting is enabled in *BfvLineCallProgressEnable*.

`CP_TYPE_CADENCE` 5

Cadence information. The cadence information is contained in *args.cadence_pattern*. Use this argument only when cadence reporting is enabled in *BfvLineCallProgressEnable*.

`args.data`

Contains the associated data corresponding to the call progress type.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

`args.cadence_pattern`

When *args.type* is `CP_TYPE_CADENCE`, the 4 elements of this array contain the high/low/high/low durations in milliseconds that make up the cadence of the detected signal. The first high and low durations might be reported as 0, which means that only a single high and low are required to describe the cadence. The function only detects signals in the range 300-640 Hz.

`args.freq_data`

When *args.type* is `CP_TYPE_FREQUENCY`, the 3 elements of this array each contain tone information consisting of a frequency in Hz and a level in cBm. The three tones together describe the detected signal.

Details

Removes data from an internal call progress buffer and returns it using *args.type* and *args.data*. The field *args.type* identifies the kind of data, and the field *args.data* identifies the associated data that corresponds to the call progress type.

The values returned in *args.data* correspond to the CS_... names defined in the header file named *btlib.h*.

This function is incorporated into the higher-level function *BfvLineOriginateCall*.

The function processes any interrupts available from the channel before checking the call progress buffer for data, so all call progress data generated by the channel is processed, stored, and available to the user.

If the call progress buffer fills up and incoming call progress data overwrites unread data, the function reinitializes the buffer, clearing the entire contents of the buffer (128 bytes) to ensure the proper chronological order of all subsequent data. Otherwise, call progress data remains in the buffer until it is read, even between calls.

When frequency reporting is enabled, the firmware analyzes tone frequencies and reports the 3 sets of frequency and level values together. The frequencies are 2-byte unsigned integers (Hz) and the levels are 2-byte signed integers (cBm). The values are returned through *BfvDataCP* in the following way: Freq 1 low byte, Freq 1 high byte, Level 1 low byte, Level 1 high byte, etc. Each byte is returned by a subsequent call with the type `CP_TYPE_FREQUENCY`.

See Also

BfvLineCallProgressEnable

Example

```
struct args_telephone args;

fp = fopen("fsk_cp_stats", "a");
fprintf(fp, "For Line # %d\n", LINE_UNIT_NUM(lp));

for (;;)
{
    BT_ZERO(args);
    if (BfvDataCP(lp, &args) == 1)
        break;
    fprintf(fp, "CP INFO: type = %x, data = %x\n",
        args.type & 0xff, args.data & 0xff);
}
```

BfvLineCallProgressDisable

Purpose

Disables call progress.

Syntax

```
void
BfvLineCallProgressDisable (lp, args)
   BTLINE                   *lp;
   struct args_telephone    *args;
```

The structure contains the following fields.

Input Fields

None

Output Field

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

This function is incorporated into the higher-level function *BfvLineOriginateCall*. See *Volume 2, Chapter 2*.

See Also

BfvLineCallProgressEnable

Example

```
BTLINE *lp
struct args_telephone args;

BT_ZERO(args);
BfvLineCallProgressDisable(lp,&args);
```

BfvLineCallProgressEnable

Purpose

Enables call progress with one of three call protocols: voice, fax, or raw.

Syntax

```
int
BfvLineCallProgressEnable  (lp, args)
   BTLINE                  *lp;
   struct args_telephone   *args;
```

The structure contains the following fields.

Input Fields

```
int call_protocol_code;
int call_mode;
unsigned report_cadence;
unsigned report_freq;
unsigned freq_report_time;
int diss_only;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

`args.call_protocol_code`

Selects a calling protocol for the channel from one of the following:

`CALL_PROTOCOL_FAX`

Selects the fax protocol. This setting requests the module to report the results from the module's call progress analysis along with the raw data. The module reports the results as soon as it establishes the fax connection or encounters a busy condition.

`CALL_PROTOCOL_RAW`

Selects a value that requests the module to report the raw high and low call progress results without performing any analysis. To use `CALL_PROTOCOL_RAW`, the application must analyze the raw call progress data.

`CALL_PROTOCOL_VOICE`

Selects the voice protocol. This setting requests the module to report the results from the module's call progress analysis along with the raw data. The module reports the results as soon as it detects a human or other answer condition.

Note: See *Appendix F, DISS - Limited Call Progress Mode*, for more information on the results from call progress analysis.

`CALL_PROTOCOL_FAX_NO_RAW`

`CALL_PROTOCOL_VOICE_NO_RAW`

Selects either the fax or voice protocol, requesting the module to report the results from the module's call progress analysis without including the raw data.

Other detected call progress results are returned after the *ced_timeout* (the length of time to wait for the called station id signal) timeout.

The *ced_timeout* value is obtained from the *BT_CPARM.CFG* configuration file or from the user-defined configuration file (see *Volume 6, Appendix A*).

`args.call_mode`

Selects one of two possible call modes, `BT_ORIGINATE` or `BT_ANSWER`, that define which call progress signals the channel expects to receive.

`BT_ORIGINATE` – Normally used; must be used to detect G2 fax tone and (with fax protocol) to play a CNG tone.

`BT_ANSWER` – Must be used to detect CNG tone; automatically enables voice protocol. In answer mode, `CALL_PROTOCOL_VOICE` and `CALL_PROTOCOL_VOICE_NO_RAW` replace `CALL_PROTOCOL_FAX` and `CALL_PROTOCOL_FAX_NO_RAW`, respectively. See *Volume 6, Appendix F, Call Progress Notes* for more information on *args.call_mode*.

`args.report_cadence`

If nonzero, enables cadence analysis and reporting. This mode is available in full call progress mode only, not in limited call progress mode (`DISS`).

Do not enable call progress programming in conjunction with this option. (See BfvLineCallProgressProgram).

This argument is meant for development and debugging purposes. When enabled, most other call progress analysis is disabled. Use of this mode is limited to a single channel on a DSP at a time.

`args.report_freq`

If nonzero, enables frequency analysis and reporting. This mode is available in full call progress mode only, not limited call progress mode (`DISS`).

Do not enable call progress programming in conjunction with this option. (See BfvLineCallProgressProgram).

This argument is intended for development and debugging purposes. When enabled, most other call progress analysis is disabled. Use of this mode is limited to a single channel on a DSP at a time.

`args.freq_report_time`

If nonzero, specifies frequency reporting interval, in milliseconds. The default is 1000.

`args.diss_only`

When set to 1, unconditionally enables limited call progress mode (DISS). In this case, all other input options are ignored. This mode must be enabled when using full-duplex speech operations, since full call progress is not compatible with full-duplex speech.

Output

Return value:

0 Call progress was successfully enabled.

<0 An error condition occurred; enabling call progress failed.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

The firmware performs sophisticated call progress functions.

The firmware automatically adapts to most international call progress signals, but applications that possess special knowledge of the dialed phone number might get faster results.

In CALL_PROTOCOL_VOICE or CALL_PROTOCOL_FAX mode, the channel continues to report the CALL_PROTOCOL_RAW output values that can be analyzed by the user-supplied function, and it also analyzes those values for meaningful patterns, such as ring-back, busy, and answer. If a final call progress value is detected (see *args.res*), call progress is halted (fax and voice modes only).

Because connection to a fax machine is the goal in fax mode, some results are suppressed until the end of the *ced_timeout* timeout. For example, if a person who answers a phone call realizes the beeping on the line is a fax machine and switches on a fax machine, fax mode suppresses the FCP_ANSWER or FCP_HUMAN result and reports FCP_ANSWER_TONE_DETECT when the fax machine is switched on.

If the remote fax machine is not detected and the *ced_timeout* timeout expires, FCP_RNGNOANS or FCP_ANSWER (or other call progress tone) is reported, depending on the conditions detected.

Ring-back (FCP_RING1, FCP_RING2) is not a final call progress value. However, ring-back is detected and reported as an intermediate value as are the CALL_PROTOCOL_RAW call progress values (where applicable).

When an application enables call progress and then plays or records speech, it provides only limited call progress (DISS mode) results. The final call progress results that are available are `BUSY1`, `ROBUSY`, `DIALTON`, `CNG`, `CED`, `CUSTOM_DIS_FREQ0`, and `CUSTOM_DIS_CAD0`. No raw call progress data are available.

Full call progress cannot be used in conjunction with *BfvCPGen*, *BfvCPGenAdv*, *BfvTonePlayBeep*, or any of the speech functions, but DISS mode can be used (see *args.diss_only*).

Do not enable call progress before placing a call using *BfvLineOriginateCall*, *BfvLineDialString* or any other means, before generating tones using *BfvTonePlay* or before beginning fax operation. Full call progress cannot be used in conjunction with *BfvCPGen*, *BfvCPGenAdv*, or *BfvTonePlayBeep*, but DISS mode can be used (see *args.diss_only*).

This function does not analyze call progress results. The application must monitor the call progress codes using *BfvDataCP*.

This function is incorporated into the higher-level function *BfvLineOriginateCall*.

In `CALL_PROTOCOL_RAW` mode, the channel reports the output of the call progress filter (a 300 – 600 Hz bandpass filter on each channel) and the time the output was in that particular state only. In this mode, the channel disables call analysis and provides raw call progress signal data directly to the user. The user then has complete control of signal interpretation.

When using full-duplex speech, *args.diss_only* should be set to 1, since full call progress is not compatible with full-duplex speech.

Note: See *Appendix F, Call Progress Notes*, for more information on the results from call progress analysis.

See Also

BfvDataCP, *BfvLineCallProgressDisable*, *BfvLineCallProgressProgram*

Example

```
BTLINE *lp;
struct args_telephone args;

BT_ZERO(args);
args.call_protocol_code = CALL_PROTOCOL_FAX;
args.call_mode = BT_ORIGINATE;
if (BfvLineCallProgressEnable(lp, &args) != 0)
    printf ("Unable to Enable Call Progress\n");
```

BfvLineCallProgressProgram

Purpose

Programs custom frequency and cadence detectors used during call progress monitoring.

Syntax

```
void
BfvLineCallProgressProgram (lp, args)
   BTLINE                  *lp;
   struct args_telephone   *args;
```

The structure contains the following fields.

Input Fields

```
unsigned template_number;
unsigned prog_freq;
unsigned prog_cadence;
unsigned diss_mode;
unsigned short frequencies[2];
unsigned duration;
int level;
unsigned short cadence_pattern[4];
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.template_number

The template number to program. In full call progress mode, the template number can be 0-7. In limited call progress mode (`DISS`), the template number must be 0.

`args.prog_freq`

If nonzero, frequency programming is in effect. Enable either this or *args.prog_cadence*. *args.frequencies*, *args.duration*, and *args.level* to program a frequency detector.

`args.prog_cadence`

If nonzero, cadence programming is in effect. You must enable either this or *args.prog_freq*. To program a cadence detector in full call progress mode, use *args.cadence_pattern*. In limited call progress mode, also use *args.frequencies* and *args.level*.

`args.diss_mode`

If nonzero, programming is in effect for the limited call progress performed during speech (`DISS`).

`args.frequencies`

An array of two frequency values in Hz (200-3900Hz). Used for programming frequency analysis in both modes and for cadence analysis in limited call progress mode (`DISS`). For limited call progress mode, only the first frequency is used. If both values are nonzero, the lower frequency must come first.

When you want to detect dual tone frequencies, the lower frequency must be the first element of the array.

`args.duration`

Minimum tone duration, in milliseconds (10ms), before a tone is recognized. Used for programming frequency analysis.

`args.level`

Minimum tone magnitude, in cBm, for a tone to be detected. Used for programming frequency analysis in both modes and for cadence analysis in limited call progress mode (`DISS`).

`args.cadence_pattern`

An array of four duration values, in milliseconds. These represent the high/low/high/low durations defining a cadence pattern for full call progress. Used for programming cadence analysis. In limited call progress mode (`DISS`), only the first two values are used. For full call progress, only signals in the range 300-640Hz are detected.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

When a tone or cadence is analyzed and found to match one of the user-programmed detectors, a call progress value of `CS_CUSTOM_FREQ0` (or 1 to 7), `CS_CUSTOM_CAD0` (or 1 to 7), `CS_CUSTOM_DIS_FREQ0`, `CS_CUSTOM_DIS_CAD0` is reported using *BfvDataCP*, based on the matching template number.

For full call progress there are 16 possible detectors (sets of analysis parameters) that can be programmed; 8 for frequency, 8 for cadence. For limited call progress (`DISS`) there is only one detector, either frequency or cadence.

Do not enable call progress programming in conjunction with the *report_freq* or *report_cadence* options of *BfvLineOrigCallDB*, or *BfvLineCallProgressEnable*.

To determine the correct parameters for programming a custom detector to detect a given cadence or a single or dual tone signal, enable call progress using the *report_cadence* or *report_freq* option of *BfvLineCallProgressEnable*.

Cadence detection programming accuracy is 15%. Two intervals are considered to be equal if: |A - B| < (A * 0.15)

Where:

- A is the interval defined for the detector.
- B is the interval measured by call progress detectors.

Cadence high intervals that are longer than the `DIALTONE` detection length are not measured correctly. The `DIALTONE` detection length is defined by the *dtone_len* parameter in the *BT_CPARM.CFG* configuration file.

For frequency detection programming, a single tone can be detected with maximum deviation of 30 Hz. If a dual tone signal has a frequency difference of < 120 Hz between the tones, the signal detector might classify the incoming signal as a single tone signal. In such cases, the single tone is the proper way to program the detector.

See Also

BfvDataCP

Example

```
BTLINE *lp;
struct args_telephone args;

/* Program custom frequency 0 for the pair of 1000 and
   1100 Hz, with a minimum level of -100 cBm and minimum
   duration of 100ms. */
BT_ZERO(args);
args.prog_freq = 1;
args.template_number = 0;
args.frequencies[0] = 1000;
args.frequencies[1] = 1100;
args.duration = 100;
args.level = -100;
BfvLineCallProgressProgram(lp,&args);

/* Program custom cadence 0 for a pattern of 200ms on,
   200ms off, 500ms on, 300ms off. */
BT_ZERO(args);
args.prog_cadence = 1;
args.template_number = 0;
args.cadence_pattern[0] = 200;
args.cadence_pattern[1] = 200;
args.cadence_pattern[2] = 500;
args.cadence_pattern[3] = 300;
BfvLineCallProgressProgram(lp,&args);

/* Enable call progress so that the previously programmed
   detectors may match signals on the line. */
BT_ZERO(args);
args.call_protocol_code = CALL_PROTOCOL_VOICE;
args.call_mode = BT_ORIGINATE;
BfvLineCallProgressEnable(lp,&args);
```

BfvToneDetectDisable

Purpose

Disables tone detection.

Syntax

```
void
BfvToneDetectDisable        (lp, args)
   BTLINE                   *lp;
   struct args_tone         *args;
```

The structure contains the following fields.

Input Fields

None

Output Field

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

Tone detection must already be enabled before calling this function (refer to the Release Notes to determine whether the release supports the tone detection feature). Any previously detected tones are placed in the tone buffer and can be retrieved with the *BfvToneGet* function.

See Also

BfvToneDetectEnable

Example

```
BTLINE *lp;
struct args_tone args;

BT_ZERO(args);
BfvToneDetectDisable(lp, &args);
```

BfvToneDetectEnable

Purpose

Enables tone detection.

Syntax

```
void
BfvToneDetectEnable          (lp, args)
   BTLINE                    *lp;
   struct args_tone          *args;
```

The structure contains the following fields.

Input Field

int *decode_flag*;

Output Field

RES *res*;

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.decode_flag

Selects DTMF or MF tone decoding and whether to report the ending of detected tones.

DTMF_12TONE	12- digit DTMF decoding
DTMF_16TONE	16-digit DTMF decoding
TONE_R1	R1 MF tones
TONE_R2_FORWARD	R2 forward MF tones
TONE_R2_BACKWARD	R2 backward MF tones

To enable the reporting of detected tone endings, logically OR this argument's value with TONE_REPORT_END.

To enable the detection of DTMF digits as RTP events (RFC_2833 digits), logically OR this argument's value with RTP_EVENTS. This is regardless of whether TONE_REPORT_END was also OR'ed.

RTP_EVENTS is only meaningful when either DTMF_12TONE or DTMF_16TONE is selected. Also, note that *rfc_2833_enabled callctrl.cfg* parameter must be configured as TRUE for RTP_EVENTS to take effect.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Tone detection must be in the disabled state when calling this function.

TONE_R1, TONE_R2, and TONE_R2_BACKWARD tones are not supported on the SR140.

The function places detected tones in a tone buffer where they can be retrieved with the *BfvToneGet* function. Normally, a value is stored in the tone buffer at the beginning of the detection of a tone. When the TONE_REPORT_END feature is used, the function also stores 0 in the tone buffer when detecting the end of a tone.

Several different tone types might be detected, but only one type of detection can be enabled at a time.

DTMF tones have a standard mapping between digits and pairs of frequencies. The following are the mappings used for MF tones. All frequencies are in Hz.

Digit	R1	R2/forward	R2/backward
0	1300/1500	1740/1860	780/660
1	700/900	1380/1500	1140/1020
2	700/1100	1380/1620	1140/900
3	900/1100	1500/1620	1020/900
4	700/1300	1380/1740	1140/780
5	900/1300	1500/1740	1020/780
6	1100/1300	1620/1740	900/780
7	700/1500	1380/1860	1140/660
8	900/1500	1500/1860	1020/660
9	1100/1500	1620/1860	900/660

Digit	R1	R2/forward	R2/backward
*	1500/1700	N/A	N/A
#	1100/1700	1860/1980	660/540
A	700/1700	1380/1980	1140/540
B	900/1700	1500/1980	1020/540
C	1300/1700	1620/1980	900/540
D	N/A	1740/1980	780/540

See Also

BfvToneGet

Example

```
BTLINE *lp;
struct args_tone args;

BT_ZERO(args);
args.decode_flag = DTMF_12TONE;
```

or

```
args.decode_flag = DTMF_12TONE | TONE_REPORT_END;
BfvToneDetectEnable(lp, &args);
```

or

```
args.decode_flag = DTMF_16TONE | RTP_EVENTS;
```

BfvToneFlush

Purpose

Discards all of the tones in the tone buffer.

Syntax

```
void
BfvToneFlush                (lp, args)
   BTLINE                   *lp;
   struct args_tone         *args;
```

The structure contains the following fields.

Input Fields

None

Output Field

RES *res;*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

See Also

BfvToneGet

Example

```
BTLINE *lp;
struct args_tone args;

BT_ZERO(args);
BfvToneFlush(lp, &args);
```

BfvToneGet

Purpose

Retrieves the next tone from the tone buffer and removes it from the buffer.

Syntax

```
int
BfvToneGet                  (lp, args)
   BTLINE                   *lp;
   struct args_tone         *args;
```

The structure contains the following fields.

Input Fields

```
long millisecs;
int tone_cp;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.millisecs

The amount of time (in milliseconds) to wait for the arrival of a tone.

args.tone_cp

If set to 1, enables checking for available call progress information in addition to tones.

Output

Return value:

Returns a detected tone as an ASCII value.

1	Indicates call progress is available for the *BfvDataCP* function to read, and no tone data is available for this function to return (if *args.tone_cp* is set to 1).
0	Indicates this function detected the end of a tone if the application enabled tone detection using the `TONE_REPORT_END` option of the *BfvToneDetectEnable* function.
<0	Indicates a timeout occurred and no tone was available, or another error condition occurred.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

See Also

BfvToneFlush, *BfvTonePeek*, *BfvToneUnget*

Example

```
BTLINE *lp;
int tone;
struct args_tone args;

BT_ZERO(args);
args.millisecs = 10000L;
if ((tone = BfvToneGet(lp, &args)) >= 0)
{
    printf("Received Tone\n");
}
```

BfvTonePeek

Purpose

Retrieves the next tone from the tone buffer without removing it.

Syntax

```
int
BfvTonePeek                 (lp, args)
   BTLINE                   *lp;
   struct args_tone         *args;
```

The structure contains the following fields.

Input Fields

int *tone_cp*;

Output Fields

RES *res*;

Modified Fields

millisecs, digit

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.tone_cp

If set to 1, enables checking for available call progress information in addition to tones.

Output

Return value:

Returns a detected tone as an ASCII value.

l	Indicates call progress is available for the *BfvDataCP* function to read, and no tone data is available for this function to return (if *args.tone_cp* is set to 1).
0	Indicates this function detected the end of a tone if the application enabled tone detection using the `TONE_REPORT_END` option of the *BfvToneDetectEnable* function.
<0	Indicates a timeout occurred and no tone was available, or another error condition occurred.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

See Also

BfvToneGet

Example

```
BTLINE *lp;
int digit;
struct args_tone args;

BT_ZERO(args);
if ((digit = BfvTonePeek(lp, &args)) < 0)
    printf("No Tones in Buffer\n");
```

BfvTonePlay

Purpose

Plays a tone for the specified time.

Syntax

```
void
BfvTonePlay                 (lp, args)
   BTLINE                   *lp;
   struct args_tone         *args;
```

The structure contains the following fields.

Input Fields

```
int digit;
long millisecs;
int tone_type;
unsigned interdigit_time;
```

Output Field

```
RES res;
```

Modified Field

decode_flag

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.digit

Designates the digit to play. The ASCII characters 0 – 9,*, #, A, B, C, and D identify the corresponding tones. See *BfvToneDetectEnable* for information about tone mappings for MF tones.

args.millisecs

The amount of time (in milliseconds) the tone plays.

`args.tone_type`

Selects the tone type.

`DTMF_12TONE`	12- digit DTMF decoding
`DTMF_16TONE`	16-digit DTMF decoding
`TONE_R1`	R1 MF tones
`TONE_R2_FORWARD`	R2 forward MF tones
`TONE_R2_BACKWARD`	R2 backward MF tones

`args.interdigit_time`

If nonzero, overrides the default interdigit time with the value specified, in ms.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Tone digits and letters must be specified as character constants (that is, ‘4’ not 4).

Generated tones are always separated by a minimum amount of silence that is specified by *tone_inter_time* (see *Volume 6, Appendix G, Country-Specific Parameter Files*).

`TONE_R1`, `TONE_R2`, and `TONE_R2_BACKWARD` tones are not supported on the SR140.

Example

```
struct args_tone args;

BT_ZERO(args);
args.digit = '4';
args.millisecs = 5000L;
args.tone_type = DTMF_TONE;
BfvTonePlay(lp, &args);
```

BfvTonePlayBeep

Purpose

Plays a single frequency tone.

Syntax

```
void
BfvTonePlayBeep              (lp, args)
   BTLINE                    *lp;
   struct args_tone          *args;
```

The structure contains the following fields.

Input Fields

```
int  toneid;
long millisecs;
int  volume;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.toneid

Indicates which of the possible tones to play.

Valid values are:

TONEID_CCITT_1 462 Hz	0
TONEID_CCITT_2 1100 Hz	1
TONEID_CCITT_3 1650 Hz	2
TONEID_CCITT_4 1850 Hz	3

`TONEID_CCITT_5`	4
2100 Hz	
`TONEID_697_HZ`	7
DTMF row 1	
`TONEID_770_HZ`	13
DTMF row 2	
`TONEID_852_HZ`	19
DTMF row 3	
`TONEID_941_HZ`	5
DTMF row 4	
`TONEID_1209_HZ`	8
DTMF col 1	
`TONEID_1336_HZ`	10
DTMF col 2	
`TONEID_1477_HZ`	12
DTMF col 3	
`TONEID_1633_HZ`	6
DTMF col 4	

`args.millisecs`

Indicates the amount of time the tone plays in milliseconds.

`args.volume`

Indicates the transmit level of the tone in dBm.

The range is –20 to 3 dBm.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The application must end signal generation before performing any other activity (for example: sending a fax, playing speech, enabling full call progress, or playing touch tones). You can use limited call progress (`DISS`).

Example

```
BTLINE *lp;
struct args_tone args;

/* Play tone 4 for a half-second at max volume */
BT_ZERO(args);
args.toneid = 4;
args.millisecs = 500L;
args.volume = 3;
BfvTonePlayBeep(lp, &args);
```

BfvToneUnget

Purpose

Puts a tone at the top of the tone buffer, so it is available for the next request to retrieve *a* tone.

Syntax

```
void
BfvToneUnget                (lp, args)
   BTLINE                   *lp;
   struct args_tone         *args;
```

The structure contains the following fields.

Input Field

```
int digit;
```

Output Field

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.digit

The ASCII representation of the digit stored in the tone buffer.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Only one tone can be returned to the top of the tone buffer. To back up more than one tone, you must write your own application-level function.

15 - Voice Play and Record

This chapter describes functions to play, record and control voice from wave or raw files, raw data buffers or an infopkt stream.

Not all functions in this chapter are fully supported on the SR140. The functions not fully supported are noted.

With the Bfv voice record and play functions, the application can:

- Open, play, and close a previously recorded prompt file.
- Record speech into an infopkt stream, a raw speech data buffer, a raw speech file, or a wave file.
- Play back speech from an infopkt stream, a raw speech data buffer, a raw speech file, or a wave file.
- Modify the volume and rate of a speech playback while it is in progress.

The voice functions allow you to write Interactive Voice Response (IVR) systems to record prompts for later playback. You can also build voice mail systems for recording and playing back messages.

TruFax® boards do not support this functionality.

Voice Play and Record Function Summary

Table 24 describes the voice play and record functions. Details about each function begin on *page 531*, in alphabetical order.

Table 24. Voice Play and Record Function Summary

Function	Purpose	Page
BfvPromptPlay	Plays phrases from a prompt file.	*531*
BfvSpeechEchoCancelControl	Allows enabling, disabling, resetting echo cancellation. Also allows routing of echo cancelled data using an alternate timeslot or accepting echo cancellation input from an alternate timeslot.	*535*
BfvSpeechModify	Enables an application to modify the volume (gain) and rate of a speech playback while it is in progress.	*538*
BfvSpeechPlay	Plays speech from the infopkt stream.	*541*
BfvSpeechPlayData	Plays raw speech data from a data buffer.	*544*
BfvSpeechPlayFile	Plays raw speech data from a file.	*551*
BfvSpeechPlayWave	Plays speech from a wave file.	*557*
BfvSpeechRecord	Retrieves the summation group number assigned to a channel.	*561*
BfvSpeechRecord	Records speech in infopkt format.	*561*
BfvSpeechRecordData	Records raw speech data into the specified buffer using the specified speech parameters.	*570*
BfvSpeechRecordFile	Records raw speech data into the specified file using the specified speech parameters.	*580*
BfvSpeechRecordWave	Records speech into the specified wave (*.wav*) file using the specified speech parameters.	*589*

BfvPromptPlay

Purpose

Plays a phrase from a prompt file, often in a sequence with other phrases.

Syntax

```
void
BfvPromptPlay                (lp, args)
   BTLINE                    *lp;
   struct args_speech        *args;
```

The structure contains the following fields.

Input Fields

```
PROMPT_MAP *prompt_map;
unsigned phrase_number;
int (*func)(BTLINE *lp, char *arg);
char *arg;
unsigned min_callback;
```

Output Fields

```
RES res;
```

Modified Fields

buf, *size*, *coding_fmt*, *rate*, *bits_per_samp*, *afe_rate*, *data_fmt*, *playf_cont*, *fname*, *ips*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.prompt_map

Pointer to the PROMPT_MAP pointer returned by the *BfvPromptOpen* function or NULL to end speech playback.

args.phrase_number

The number of the phrase within the prompt file you want the function to play.

`args.func`

Pointer to a user-supplied function called during speech playback. `NULL` disables this feature.

The *args.func* argument is called as `(*args.func)(lp,args.arg)`.

The *lp* variable contains the pointer to the line structure; *args.arg* contains the supplied user-defined argument.

Returns an integer:

0 Maintains speech playback.

1 Aborts speech playback.

`args.arg`

Argument to *args.func*. Can be `NULL`.

`args.min_callback`

The maximum time (in milliseconds) that the Bfv API allows to elapse between calls to the user-supplied callback function (*args.func*). Zero (0) indicates that the calling of the callback function depends on the activity taking place on the channel.

Note: Setting this parameter to a nonzero value smaller than one second might have an adverse effect on the system CPU usage. The smaller the value, the greater the effect on CPU usage.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Speech playback continues uninterrupted between phrases as long as the underlying speech format does not change. Continuity of speech playback eliminates clicks and delays that result from stopping and restarting speech playback. After the application has played the last phrase in a sequence, it must call the function with a *prompt_map* value of `NULL` to specify the end of speech.

The phrases in the prompt file are infopkt sequences with an `END_OF_SPEECH` infopkt at the end. This function plays `SPEECH` infopkts and follows indirect (`INDIR`) infopkts (places infopkts identified by the `INDIR` pointer into the stream).

`INFOPKT_SPEECH_PARAMETERS` infopkts typically precede `SPEECH` infopkts.

Playing continues until the function reaches the end of the last phrase or the user-supplied function returns a nonzero value.

If the user function stops playback, the *BfvPromptPlay* function discards data remaining in the driver and firmware buffers.

See *Table 25* through *Table 27*, starting on *page 569*, for information about valid speech settings.

Checks to ensure that the line state is set to `CONNECTED` when the function starts and during the buffer read loops.

Speech playback terminates when the line state changes to `IDLE`.

The user-supplied function must not call any function that causes a delay, such as waiting for a `DTMF` tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing `DTMF` tone detection must enable detection before beginning the operation that uses the user-supplied function (for example, speech playback).

If you set a nonzero value for *args.min_callback*, the Bfv API makes sure that it calls the *args.func* user-supplied callback function at least every *args.min_callback* milliseconds. If you set *args.min_callback* to zero, callback function calling depends on activity taking place on the channel.

See Also

BfvSpeechModify, *BfvSpeechPlay*, *BfvPromptOpen*

Example

```
BTLINE *lp;
struct args_infopkt args_info;
struct args_speech args_speech;
PROMPT_MAP *prompt_map;
...

BT_ZERO (args_info);
args_info.name = “prompt.ips”;
args_info.fmode = “r”;
prompt_map = BfvPromptOpen (&args_info);

BT_ZERO (args_speech);
args_speech.prompt_map = prompt_map;
args_speech.phrase_number = 2;
args_speech.func = play_func;
BfvPromptPlay (lp, &args_speech);
...

int play_func (lp, arg)
BTLINE *lp
char *arg;
{
    ...
}
```

BfvSpeechEchoCancelControl

Purpose

Allows enabling, disabling, or resetting echo cancellation.

Syntax

```
void
BfvSpeechEchoCancelControl (lp, args)
   BTLINE                   *lp;
   struct args_speech       *args;
```

The structure contains the following fields.

Input Fields

```
int echoc_op;
int output_select;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.echoc_op

Selects the operation to perform. Valid values include:

ECHOC_OP_ENABLE

Enable echo cancellation.

ECHOC_OP_DISABLE

Disable echo cancellation.

ECHOC_OP_RESET

Reset echo cancellation.

`ECHOC_OP_ALT_INPUT_ENABLE`

Enable acceptance of input for echo cancellation from an alternate timeslot. The timeslot is timeslot 1 on the current channel.

`ECHOC_OP_ALT_OUTPUT_SELECT`

Select routing of echo cancelled data. The selection is specified by *args.output_select*.

`args.output_select`

When *args.echoc_op* is `ECHOC_OP_ALT_OUTPUT_SELECT`, selects the routing method of echo cancelled data. Valid values include:

`SPEECH_RECORD_TO_HOST_TSLOT_DEF` 0x01

Send recorded data to both the host and the alternate timeslot. The timeslot is timeslot 1 on the current channel.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 does not support this function.

This function controls echo cancellation. Use the function for:

- Enabling, disabling, resetting echo cancellation.
- Routing of echo cancelled data using alternate timeslot or accepting echo cancellation input from an alternate timeslot.

BfvSpeechEchoCancelControl configuration persists, even after *BfvLineReset* or program exit. For example, if you turn off echo cancellation and exit your program and restart, echo cancellation will still be off.

When using an alternate timeslot for echo cancelled input or output, the timeslot is timeslot 1 on the current channel. Use *BfvCallSWConnect* to establish an appropriate half-duplex connection between the ultimate input or output stream and the alternate timeslot.

This timeslot is *port_class* `CALL_SW_PORT_CHANNEL_DEF`. *port_unit* is equal to the logical channel number on the module, stream 0, slot 1. The first channel on each module is logical channel 2, the second channel is logical channel 3, and so on. You can use the `LINE_DEST_ADDR` macro to determine the logical channel number.

Example

```
/* Enable echo cancellation */
struct args_speech args;

BT_ZERO(args);
args.echoc_op = ECHOC_OP_ENABLE;
BfvSpeechEchoCancelControl (lp,&args);
```

BfvSpeechModify

Purpose

Enables an application to modify the volume (gain) of speech playback while it is in progress.

Syntax

```
void
BfvSpeechModify              (lp, args)
   BTLINE                    *lp;
   struct args_speech        *args;
```

The structure contains the following fields.

Input Fields

```
int modification;
int value;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.modification

An integer value that represents the type of modification to perform.

SPCH_MOD_GAIN_UP 2

Increase volume

SPCH_MOD_GAIN_DOWN 3

Decrease volume

SPCH_MOD_GAIN_SET 6

Set volume.

args.value

Parameter used when setting volume using SPCH_MOD_GAIN_SET .

Valid values for SPCH_MOD_GAIN_SET are:

–18 dB	–3
–12 dB	–2
–6 dB	–1
0 dB	0
+6 dB	1
+12 dB	2
+18 dB	3

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

With each application of the SPCH_MOD_GAIN_SET modification value, the volume increases or decreases by a factor of 2. Maximum increase is 100% above the original gain, and maximum decrease is 50% below the original gain.

Applications can call this function from within the user-defined function.

Using this function, applications can enable users to adjust the volume of a recording during play by pressing a specified key (DTMF tone) on their telephone keypad.

Applications can call this function multiple times.

See Also

BfvSpeechPlay, *BfvSpeechPlayData*, *BfvSpeechPlayFile*

Example

```
BTLINE *lp;
struct args_speech args_speech;
...

BT_ZERO (args_speech);
args_speech.modification = SPCH_MOD_GAIN_UP;
```

```
BfvSpeechModify (lp, &args_speech);
```

BfvSpeechPlay

Purpose

Plays speech from the infopkt stream.

Syntax

```
void
BfvSpeechPlay                (lp, args)
   BTLINE                    *lp;
   struct args_speech        *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *ips;
int (*func)(BTLINE *lp, char *arg);
char *arg;
unsigned min_callback;
```

Output Fields

```
unsigned bytes_processed;
RES res;
```

Modified Fields

buf, *size*, *coding_fmt*, *rate*, *bits_per_samp*, *afe_rate*, *data_fmt*, *playf_cont*, *fname*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.ips

Pointer to an infopkt stream that is opened for reading.

`args.func`

Pointer to a user-supplied function called during speech playback. `NULL` disables this feature.

The *args.func* argument is called as `(*args.func)(lp,args.arg)`.

The *lp* variable contains the pointer to the line structure; *args.arg* contains the supplied user-defined argument.

Returns an integer:

0 Maintains speech playback.

1 Aborts speech playback.

`args.arg`

Argument to *args.func*. Can be `NULL`.

`args.min_callback`

The maximum time (in milliseconds) that the Bfv API allows to elapse between calls to the user-supplied callback function (*args.func*). Zero (0) indicates that the calling of the callback function depends on the activity taking place on the channel.

Note: Setting this parameter to a nonzero value smaller than one second might have an adverse effect on the system CPU usage. The smaller the value, the greater the effect on CPU usage.

Output

Return value: None.

`args.bytes_processed`

The number of bytes of raw speech data played. This field is only valid when speech playback has terminated with a non-error return.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function plays SPEECH infopkts and places infopkts identified by the INDIR pointer into the stream. All other infopkt types are ignored.

The first non-INDIR infopkt must be INFOPKT_SPEECH_PARAMETERS that sets the bit rate.

See *Table 25* through *Table 27*, starting on *page 569*, for information about valid speech settings.

Playing continues until one of the following conditions is met:

- The infopkt stream is finished.
- The user-supplied function returns a nonzero value.
- The function encounters an END_OF_SPEECH infopkt with a mode parameter value of 0.

If the user function stops playback, the *BfvSpeechPlay* function discards data remaining in the driver and firmware buffers.

The function checks to ensure that the line state is set to CONNECTED when the function starts and during the buffer read loops.

Speech playback terminates when the line state changes to IDLE.

The user-supplied function must not call any function that causes a delay, such as waiting for a DTMF tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing DTMF tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

If you set a nonzero value for *args.min_callback*, the Bfv API makes sure that it calls the *args.func* user-supplied callback function at least every *args.min_callback* milliseconds. If you set *args.min_callback* to zero, callback function calling depends on activity taking place on the channel.

See Also

BfvSpeechPlayData, *BfvSpeechPlayFile*

Example

See applications in the *boston/bfv.api/app.src* sample application directory.

BfvSpeechPlayData

Purpose

Plays raw speech data from a data buffer.

Syntax

```
void
BfvSpeechPlayData          (lp, args)
   BTLINE                  *lp;
   struct args_speech      *args;
```

The structure contains the following fields.

Input Fields

```
unsigned char *buf;
unsigned size;
unsigned coding_fmt;
unsigned rate;
unsigned bits_per_samp;
unsigned afe_rate;
unsigned data_fmt;
int (*func)(BTLINE *lp, char *arg);
char *arg;
int playf_cont;
unsigned min_callback;
```

Output Fields

```
unsigned bytes_processed;
RES *res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.buf

Pointer to a data buffer that contains the speech data.

args.size

The size, in bytes, of the user-allocated data buffer.

args.coding_fmt

Coding format of the data. Valid values are:

CODE_ADPCM	1
OKI ADPCM	
CODE_PCM_ULAW	2
μ-Law	
CODE_PCM_ALAW	3
A-Law	
CODE_G726	17
G.726	
CODE_GSM_610	14
Microsoft GSM 6.10	
CODE_LINEAR	4
Linear	

args.rate

The compressed sample rate for playback. If the value is RATE_STOP, speech playback is stopped. Valid values are:

RATE_STOP	
Stop playback	
RATE_5300	9
5,300 samples/sec	
RATE_6000	0
6,000 samples/sec	
RATE_8000	1
8,000 samples/sec	
RATE_9600	7
9,600 samples/sec	
RATE_11000	8
11,000 samples/sec	
RATE_12300	12
12,300 samples/sec	
RATE_13000	11
13,000 samples/sec	

`RATE_16000` 6

16,000 samples/sec

`RATE_20000` 2

20,000 samples/sec

`RATE_24000` 3

24,000 samples/sec

`RATE_28000` 4

28,000 samples/sec

`RATE_32000` 5

32,000 samples/sec

`args.bits_per_samp`

Number of bits per sample. Valid values are:

`BITS_1` 0

1 bit/sample

`BITS_2` 7

2 bit/sample

`BITS_3` 1

3 bit/sample

`BITS_4` 2

4 bits/sample

`BITS_5` 8

5 bit/sample

`BITS_8` 3

8 bits/sample

`BITS_16` 5

16 bits/sample

`args.afe_rate`

AFE sample rate. Valid values are:

`AFE_8000` 0

8000 samples/sec

`args.data_fmt`

Data format specification. Value at present is:

`SPCH_MSB` 0

MSB is first

`args.func`

Pointer to a user-supplied function called during speech playback. `NULL` disables this feature.

The *args.func* argument is called as `(*args.func)(lp,args.arg)`.

The *lp* variable contains the pointer to the line structure; *args.arg* contains the supplied user-defined argument.

Returns an integer:

0 Maintains speech playback.

1 Aborts speech playback.

`args.arg`

Argument to *args.func*. Can be `NULL`.

`args.playf_cont`

If set to 1, this function does not terminate speech playback at end of file.

`args.min_callback`

The maximum time (in milliseconds) that the Bfv API allows to elapse between calls to the user-supplied callback function (*args.func*). Zero (0) indicates that the calling of the callback function depends on the activity taking place on the channel.

Note: Setting this parameter to a nonzero value smaller than one second might have an adverse effect on the system CPU usage. The smaller the value, the greater the effect on CPU usage.

Output

Return value: None.

`args.bytes_processed`

The number of bytes of raw speech data played. This field is only valid when speech playback has terminated with a non-error return.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 supports only CODE_PCM_ULAW and CODE_PCM_ALAW coding formats, sampled at RATE_8000 samples per second with BITS_8 per sample.

This function plays raw speech data from the specified data buffer, using the specified speech parameters.

See *Table 25* through *Table 27*, starting on *page 569*, for information about valid speech settings.

Playing continues until one of the following conditions is met:

- The value of rate is `RATE_STOP`. All data to be played has been supplied and playback continues until end-of-data.
- The user-supplied function returns a nonzero value. Playback is immediately stopped.

If the user function stops playback, the *BfvSpeechPlayData* function discards data remaining in the driver and firmware buffers. The value of rate can be changed between subsequent calls to *BfvSpeechPlayData* without halting playback.

The function checks to ensure that the line state is set to `CONNECTED` when the function starts and during the buffer read loops.

Speech playback terminates when the line state changes to `IDLE`.

Employing the user-supplied function is the only way to immediately stop speech playback; a rate value of `RATE_STOP` only indicates that all data has been supplied.

The user-supplied function must not call any function that causes a delay, such as waiting for a `DTMF` tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing `DTMF` tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

If you set a nonzero value for *args.min_callback*, the Bfv API makes sure that it calls the *args.func* user-supplied callback function at least every *args.min_callback* milliseconds. If you set *args.min_callback* to zero, callback function calling depends on activity taking place on the channel.

Using *args.playf_cont* helps prevent noticeable transitions when making subsequent calls to *BfvSpeechPlayData* or *BfvSpeechPlayFile* by preventing speech playback from terminating. Terminate the speech playback using a later call to a speech playback function.

See Also

BfvSpeechModify, *BfvSpeechPlay*, *BfvSpeechPlayFile*

Example

```
BTLINE *lp;
unsigned char buf[1024];
int n;
int my_user_func();
struct args_speech args;

/* Read and play from a file until done or user function indicates to stop */
while ((n = read_file(buf, sizeof(buf)) > 0)
{
    BT_ZERO(args);
    args.buf = buf;
    args.size = n;
    args.coding_fmt = CODE_ADPCM;
    args.rate = RATE_8000;
    args.bits_per_samp = BITS_4;
    args.afe_rate = AFE_8000;
    args.data_fmt = SPCH_MSB;
    args.func = my_user_func;
    args.arg = NULL;
    BfvSpeechPlayData(lp, &args);
    if (args.res.status != BT_STATUS_OK)
        return(-1);
}
BT_ZERO(args);
args.buf = NULL;
args.size = 0;
args.coding_fmt = CODE_ADPCM;
args.rate = RATE_STOP;
args.func = my_user_func;
args.arg = NULL;
BfvSpeechPlayData(lp, &args);
if (args.res.status != BT_STATUS_OK)
    return(-1);
```

BfvSpeechPlayFile

Purpose

Plays raw speech data from a file.

Syntax

```
void
BfvSpeechPlayFile          (lp, args)
   BTLINE                  *lp;
   struct args_speech      *args;
```

The structure contains the following fields.

Input Fields

```
char fname;
unsigned char *buf;
unsigned size;
unsigned coding_fmt;
unsigned rate;
unsigned bits_per_samp;
unsigned afe_rate;
unsigned data_fmt;
int (*func)(BTLINE *lp, char *arg);
char *arg;
int playf_cont;
unsigned min_callback;
```

Output Fields

```
unsigned bytes_processed;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fname

The name of the file that contains the speech data to play.

args.buf

Pointer to a data buffer that contains the speech data.

`args.size`

The size, in bytes, of the user-allocated data buffer.

`args.coding_fmt`

Coding format of the data. Valid values are:

`CODE_ADPCM`	1
OKI ADPCM	
`CODE_PCM_ULAW`	2
µ-Law	
`CODE_PCM_ALAW`	3
A-Law	
`CODE_G726`	17
G.726	
`CODE_GSM_610`	14
Microsoft GSM 6.10	
`CODE_LINEAR`	4
Linear	

`args.rate`

The compressed sample rate for playback. If the value is `RATE_STOP`, speech playback is stopped. Valid values are:

`RATE_STOP`	
Stop playback	
`RATE_5300`	9
5,300 samples/sec	
`RATE_6000`	0
6,000 samples/sec	
`RATE_8000`	1
8,000 samples/sec	
`RATE_9600`	7
9,600 samples/sec	
`RATE_11000`	8
11,000 samples/sec	
`RATE_12300`	12
12,300 samples/sec	

`RATE_13000` 11

13,000 samples/sec

`RATE_16000` 6

16,000 samples/sec

`RATE_20000` 2

20,000 samples/sec

`RATE_24000` 3

24,000 samples/sec

`RATE_28000` 4

28,000 samples/sec

`RATE_32000` 5

32,000 samples/sec

`args.bits_per_samp`

Number of bits per sample. Valid values are:

`BITS_1` 0

1 bit/sample

`BITS_2` 7

2 bit/sample

`BITS_3` 1

3 bit/sample

`BITS_4` 2

4 bits/sample

`BITS_5` 8

5 bit/sample

`BITS_8` 3

8 bits/sample

`BITS_16` 5

16 bits/sample

`args.afe_rate`

AFE sample rate. Valid values are:

`AFE_8000` 0

8000 samples/sec

`args.data_fmt`

Data format specification. Value at present is:

`SPCH_MSB` 0

MSB is first

`args.func`

Pointer to a user-supplied function called during speech playback. `NULL` disables this feature.

The *args.func* argument is called as `(*args.func)(lp,args.arg)`.

The *lp* variable contains the pointer to the line structure; *args.arg* contains the supplied user-defined argument.

Returns an integer:

0 Maintains speech playback.

1 Aborts speech playback.

`args.arg`

Argument to *args.func*. Can be `NULL`.

`args.playf_cont`

If set to 1, this function does not terminate speech playback at end of file.

`args.min_callback`

The maximum time (in milliseconds) that the Bfv API allows to elapse between calls to the user-supplied callback function (*args.func*). Zero (0) indicates that the calling of the callback function depends on the activity taking place on the channel.

Note: Setting this parameter to a nonzero value smaller than one second might have an adverse effect on the system CPU usage. The smaller the value, the greater the effect on CPU usage.

Output

Return value: None.

`args.bytes_processed`

The number of bytes of raw speech data played. This field is only valid when speech playback has terminated with a non-error return.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 supports only CODE_PCM_ULAW and CODE_PCM_ALAW coding formats, sampled at RATE_8000 samples per second with BITS_8 per sample.

This function plays raw speech data from the specified file, using the specified parameters.

See *Table 25* through *Table 27*, starting on *page 569*, for information about valid speech settings.

The value of rate can be changed between subsequent calls to *BfvSpeechPlayFile* without halting playback.

Playing continues until one of the following conditions is met:

- The file is finished.
- The user-supplied function returns a nonzero value.

If the user function stops playback, the *BfvSpeechPlayFile* function discards data remaining in the driver and firmware buffers.

The function checks to ensure that the line state is set to `CONNECTED` when the function starts and during the buffer read loops.

Speech playback terminates when the line state changes to `IDLE`.

Employing the user-supplied function is the only way to immediately abort speech playback.

The user-supplied function must not call any function that causes a delay, such as waiting for a `DTMF` tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing `DTMF` tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

If you set a nonzero value for *args.min_callback*, the Bfv API makes sure that it calls the *args.func* user-supplied callback function at least every *args.min_callback* milliseconds. If you set *args.min_callback* to zero, callback function calling depends on activity taking place on the channel.

Using *args.playf_cont* helps prevent noticeable transitions when making subsequent calls to *BfvSpeechPlayFile* or *BfvSpeechPlayData* by preventing speech playback from terminating. Terminate the speech playback using a later call to a speech playback function.

See Also

BfvSpeechModify, *BfvSpeechPlay*, *BfvSpeechPlayData*

Example

```
BTLINE *lp;
char *fname;
int my_user_func();
struct args_speech args;

/* Play from a file until done or user function indicates
   to stop */
BT_ZERO(args);
args.fname = fname;
args.coding_fmt = CODE_ADPCM;
args.rate = RATE_8000;
args.bits_per_samp = BITS_4;
args.afe_rate = AFE_8000;
args.data_fmt = SPCH_MSB;
args.func = my_user_func;
args.arg = NULL;
BfvSpeechPlayFile(lp, &args);
```

BfvSpeechPlayWave

Purpose

Plays speech from a wave file.

Syntax

```
void
BfvSpeechPlayWave          (lp, args)
   BTLINE                  *lp;
   struct args_speech      *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
int (*func)(BTLINE *lp, char *arg);
char *arg;
int playf_cont;
unsigned min_callback;
```

Output Fields

```
unsigned bytes_processed;
RES res;
```

Modified Fields

buf, *size*, *coding_fmt*, *rate*, *bits_per_samp*, *afe_rate*, *data_fmt*

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fname

Name of the wave file to play.

`args.func`

Pointer to a user-supplied function called during speech playback. `NULL` disables this feature.

The *args.func* argument is called as `(*args.func)(lp,args.arg)`.

The *lp* variable contains the pointer to the line structure; *args.arg* contains the supplied user-defined argument.

Returns an integer:

0 Maintains speech playback.

1 Aborts speech playback.

`args.arg`

Argument to *args.func*. Can be `NULL`.

`args.playf_cont`

If set to 1, this function does not terminate speech playback at end of file.

`args.min_callback`

The maximum time (in milliseconds) that the Bfv API allows to elapse between calls to the user-supplied callback function (*args.func*). Zero (0) indicates that the calling of the callback function depends on the activity taking place on the channel.

Note: Setting this parameter to a nonzero value smaller than one second might have an adverse effect on the system CPU usage. The smaller the value, the greater the effect on CPU usage.

Output

Return value: None.

`args.bytes_processed`

The number of bytes of raw speech data played. This field is only valid when speech playback has terminated with a non-error return.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 supports only u-Law, A-Law coding formats, sampled at 8Khz with 8 bits per sample.

This function plays speech from the specified wave (*.wav*) file.

See *Table 25* through *Table 27*, starting on *page 569*, for information about valid speech settings. Because some format or rate conversions might be done at host level, the possible formats and rates might not match the native capabilities of the firmware (see *Table 25, Voice Encoding Settings*, on *page 568*).

Only wave files containing data recorded in one of the following combinations of coding format, rate, and bits per sample can be played:

Coding format = µ-Law	sample rate = 8KHz, 11KHz
	bits per sample = 8
Coding format = A-Law	sample rate = 8KHz, 11KHz
	bits per sample = 8
Coding format = Linear	sample rate = 8KHz, 11KHz
	bits per sample = 8, 16

Playing continues until one of the following conditions is met:

- The file comes to an end.
- The user-supplied function returns a nonzero value.

If the user function stops playback, the *BfvSpeechPlayWave* function discards data remaining in the driver and firmware buffers.

The function checks to ensure that the line state is set to `CONNECTED` when the function starts and during the buffer read loops.

Speech playback terminates when the line state changes to `IDLE`.

Employing the user-supplied function is the only way to immediately abort speech playback.

The user-supplied function must not call any function that causes a delay, such as waiting for a `DTMF` tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing `DTMF` tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

If you set a nonzero value for *args.min_callback*, the Bfv API makes sure that it calls the *args.func* user-supplied callback function at least every *args.min_callback* milliseconds. If you set *args.min_callback* to zero, callback function calling depends on activity taking place on the channel.

Using *args.playf_cont* helps prevent noticeable transitions when making subsequent calls to another speech playback function by preventing speech playback from terminating. Terminate the speech playback using a later call to a speech playback function.

Note: Stereo wave files are not supported.

See Also

BfvSpeechRecordWave

Example

```
BTLINE *lp;
char *fname;
int my_user_func();
struct args_speech args;

/* Play from a wave file until finished or user function
   indicates to stop */
BT_ZERO(args);
args.fname = fname;
args.func = my_user_func;
args.arg = NULL;
BfvSpeechPlayWave(lp, &args);
```

BfvSpeechRecord

Purpose

Records speech in infopkt format.

Syntax

```
void
BfvSpeechRecord              (lp, args)
   BTLINE                    *lp;
   struct args_speech        *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *ips;
unsigned char *buf;
unsigned size;
unsigned coding_fmt;
unsigned rate;
unsigned bits_per_samp;
unsigned afe_rate;
unsigned data_fmt;
long timeout;
long silence_timeout;
int s_compr;
int (*func)(BTLINE *lp, char *arg);
char *arg;
int beep;
unsigned min_callback;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.ips

Pointer to the infopkt where speech data is written.

args.buf

Pointer to the user-allocated data buffer that holds the received data. The data buffer must be at least 1024 bytes in size.

args.size

The size, in bytes, of the user-allocated data buffer.

args.coding_fmt

Coding format of the data. Valid values are:

`CODE_ADPCM`	1
OKI ADPCM	
`CODE_PCM_ULAW`	2
μ-Law	
`CODE_PCM_ALAW`	3
A-Law	
`CODE_G726`	17
G.726	
`CODE_GSM_610`	14
Microsoft GSM 6.10	
`CODE_LINEAR`	4
Linear	

`args.rate`

The compressed sample rate for playback. If the value is `RATE_STOP`, speech playback is stopped. Valid values are:

`RATE_STOP`	
Stop playback	
`RATE_5300`	9
5,300 samples/sec	
`RATE_6000`	0
6,000 samples/sec	
`RATE_8000`	1
8,000 samples/sec	
`RATE_9600`	7
9,600 samples/sec	
`RATE_11000`	8
11,000 samples/sec	
`RATE_12300`	12
12,300 samples/sec	
`RATE_13000`	11
13,000 samples/sec	
`RATE_16000`	6
16,000 samples/sec	
`RATE_20000`	2
20,000 samples/sec	
`RATE_24000`	3
24,000 samples/sec	
`RATE_28000`	4
28,000 samples/sec	
`RATE_32000`	5
32,000 samples/sec	

`args.bits_per_samp`

Number of bits per sample. Valid values are:

`BITS_1` 0
: 1 bit/sample

`BITS_2` 7
: 2 bit/sample

`BITS_3` 1
: 3 bit/sample

`BITS_4` 2
: 4 bits/sample

`BITS_5` 8
: 5 bit/sample

`BITS_8` 3
: 8 bits/sample

`BITS_16` 5
: 16 bits/sample

`args.afe_rate`

AFE sample rate. Valid values are:

`AFE_8000` 0
: 8000 samples/sec

`args.data_fmt`

Data format specification. Value at present is:

`SPCH_MSB` 0
: MSB is first

`args.timeout`

The total time (in milliseconds) to record. Zero (0) indicates that recording continues unaffected by a time limit.

`args.silence_timeout`

Stops recording when silence is detected for this amount of time (in milliseconds). Zero (0) indicates that recording continues unaffected by a period of silence. Maximum value is 5 * 65535.

`args.s_compr`

Indicates if the function removes silence. A value of 1 enables silence removal, 0 disables. Do not use this parameter during full-duplex recording and playback or if *args.asr_mode* is in use. Use in these cases.

`args.func`

Pointer to a user-supplied function called during speech recording. `NULL` disables this feature.

The *args.func* argument is called as `(*args.func)(lp,args.arg)`.

The *lp* variable contains the pointer to the line structure; *arg* contains the supplied user-defined argument.

Returns an integer:

0 Maintains speech recording.

1 Aborts speech recording.

`args.arg`

Argument to *args.func*. Can be `NULL`.

`args.beep`

When nonzero, a beep will be played prior to record starting. If the channel is already performing playback, then the beep will not be played.

By default the beep tone will be played for 500 milliseconds at 500 Hz.

The behavior is controlled by the user configuration file *record_beep_dur* and *record_beep_freq* parameters (*see Volume 6, Appendix A*).

`args.min_callback`

The maximum time (in milliseconds) that the Bfv API allows to elapse between calls to the user-supplied callback function (*args.func*). Zero (0) indicates that the calling of the callback function depends on the activity taking place on the channel.

Note: Setting this parameter to a nonzero value smaller than one second might have an adverse effect on the system CPU usage. The smaller the value, the greater the effect on CPU usage.

Output

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

The SR140 supports only CODE_PCM_ULAW and CODE_PCM_ALAW coding formats, sampled at RATE_8000 samples per second with BITS_8 per sample. The TR1034 Digital boards do not support Linear coding formats.

Not all combinations of *args.coding_fmt*, *args.rate*, and *args.bits_per_samp* are valid (see *Table 25, Voice Encoding Settings*, on *page 568*).

Recording stops at the first occurrence of one of the following:

- The value of *args.rate* is `RATE_STOP`
- Silence timeout
- Overall timeout
- The user-supplied integer function returns nonzero
- The line state changes to `IDLE`

After recording stops, the function stores the data remaining in the channel and driver buffers in the supplied infopkt stream, and clears the channel and driver buffers.

When recording is stopped because the value of rate is `RATE_STOP`, the driver buffer might still contain speech data. To receive the complete set of speech data, the application must call *BfvSpeechRecord* repeatedly until the return value is 0.

The *args.timeout* and *args.silence_timeout* arguments only have effect when recording is first begun. Values for these arguments in subsequent calls to this function for the same recording are ignored. The *args.silence_timeout* argument has no effect during summation recording.

The argument *args.func* can be `NULL`, or it can point to a user-supplied function that determines when to stop recording.

The user-supplied function is a flexible and powerful tool and can be used to:

- Detect the end of recording caused by an event. Events include:
  - Detection of a `DTMF` tone
  - Detection of a particular call progress result
  - An external trigger such as the keyboard
- Report information during recording.

The user-supplied function must not call any function that causes a delay, such as waiting for a `DTMF` tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing `DTMF` tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

If you set a nonzero value for *args.min_callback*, the Bfv API makes sure that it calls the *args.func* user-supplied callback function at least every *args.min_callback* milliseconds. If you set *args.min_callback* to zero, callback function calling depends on activity taking place on the channel.

During any of the following conditions, the user callback function might not be called regularly unless there is some activity destined for the application session. Alerts sent by *BfvLineAlert* (which can be cancelled by the application) can create such activity.

- When silence compression is enabled using *args.s_compr* and silence is detected.
- If speech data has been routed to an alternate timeslot using *BfvSpeechEchoCancelControl*.

Table 25 specifies the valid settings that applications can use to record and to playback speech. These settings include values for:

- Coding format of the data (`CODE_...`)
- Number of bits per sample (`BITS_...`). `BITS_...` does not apply to frame-based coders (`GSM_610`).
- Compressed sample rate for playback (`RATE_...`)

Table 25. Voice Encoding Settings

CODE_	BITS_	RATE_	Notes
ADPCM	4	6,000	24 kbps ADPCM
ADPCM	4	8,000	32 kbps ADPCM
PCM_ULAW	8	6,000	48 kbps µ_law
PCM_ULAW	8	8,000	64 kbps µ_law
PCM_ULAW	8	11,000	88 kbps µ_law
PCM_ALAW	8	6,000	48 kbps a_law
PCM_ALAW	8	8,000	64 kbps a_law
PCM_ALAW	8	11,000	88 kbps a_law
G726	2	8,000	16 kbps G.726
G726	3	8,000	24 kbps G.726
G726	4	8,000	32 kbps G.726
G726	5	8,000	40 kbps G.726
GSM_610	Any	13,000	13 kbps Microsoft GSM 6.10
LINEAR	16	8,000	128 kbps Linear 16-bit

See Also

BfvSpeechRecord, *BfvSpeechRecordData*, *BfvSpeechRecordFile*, *BfvSpeechRecordWave*, `LINE_HAS_CAP (lp, cap)`, `LINE_VAD_STATE (lp)`, `LINE_VAD_BYTES_PROCESSED (lp)`

Example

```
BTLINE *lp;
struct args_speech args_speech;
struct args_infopkt args_infopkt;
struct infopkt_stream *ips;
...

BT_ZERO (args_infopkt);
args_infopkt.fname = “record.ips”;
args_info.fmode = “w”;
ips = BfvInfopktOpen (&args_infopkt);

BT_ZERO (args_speech);
args_speech.ips = ips;
args_speech.coding_fmt = CODE_PCM_ULAW;
args_speech.rate = RATE_8000;
args_speech.bits_per_samp = BITS_8;
args_speech.afe_rate = AFE_8000;
args_speech.timeout = 10000L;
args_speech.silence_timeout = 0L;
args_speech.s_compr =1;
args_speech.func = play_func;
BfvSpeechRecord (lp, &args_speech);
...

int play_func (lp, arg)
BTLINE *lp
char *arg;
{
    ...
}
```

BfvSpeechRecordData

Purpose

Records raw speech data into the specified buffer using the specified speech parameters.

Syntax

```
unsigned int
BfvSpeechRecordData          (lp, args)
   BTLINE                    *lp;
   struct args_speech        *args;
```

The structure contains the following fields.

Input Fields

```
unsigned char *buf;
unsigned size;
unsigned coding_fmt;
unsigned rate;
unsigned bits_per_samp;
unsigned afe_rate;
unsigned data_fmt;
long timeout;
long silence_timeout;
int s_compr;
int (*func)(BTLINE *lp, char *arg);
char *arg;
int vad;
int vad_thresh;
unsigned vad_passthru;
int asr_mode;
int beep;
unsigned min_callback;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

`args.buf`

Pointer to the user-allocated data buffer that holds the received data. The data buffer must be at least 1024 bytes in size.

`args.size`

The size, in bytes, of the user-allocated data buffer.

`args.coding_fmt`

Coding format of the data. Valid values are:

`CODE_ADPCM`	1
OKI ADPCM	
`CODE_PCM_ULAW`	2
μ-Law	
`CODE_PCM_ALAW`	3
A-Law	
`CODE_G726`	17
G.726	
`CODE_GSM_610`	14
Microsoft GSM 6.10	
`CODE_LINEAR`	4
Linear	

`args.rate`

The compressed sample rate for playback. The lower rates are more space efficient but produce poorer speech quality. If the value is `RATE_STOP`, speech playback is stopped. Valid values are:

`RATE_STOP`	
Stop playback	
`RATE_5300`	9
5,300 samples/sec	
`RATE_6000`	0
6,000 samples/sec	
`RATE_8000`	1
8,000 samples/sec	

`RATE_9600` 7

9,600 samples/sec

`RATE_11000` 8

11,000 samples/sec

`RATE_12300` 12

12,300 samples/sec

`RATE_13000` 11

13,000 samples/sec

`RATE_16000` 6

16,000 samples/sec

`RATE_20000` 2

20,000 samples/sec

`RATE_24000` 3

24,000 samples/sec

`RATE_28000` 4

28,000 samples/sec

`RATE_32000` 5

32,000 samples/sec

`args.bits_per_samp`

Number of bits per sample. Valid values are:

`BITS_1` 0

1 bit/sample

`BITS_2` 7

2 bit/sample

`BITS_3` 1

3 bit/sample

`BITS_4` 2

4 bits/sample

`BITS_5` 8

5 bit/sample

`BITS_8` 3

8 bits/sample

`BITS_16` 5

16 bits/sample

`args.afe_rate`

AFE sample rate. Valid values are:

`AFE_8000` 0

8000 samples/sec

`args.data_fmt`

Data format specification. Value at present is:

`SPCH_MSB` 0

MSB is first

`args.timeout`

If nonzero, recording terminates after no more than *args.timeout* milliseconds elapse.

`args.silence_timeout`

Stops recording when silence is detected for this amount of time (in milliseconds). Zero (0) indicates that recording continues unaffected by a period of silence. Maximum value is 5 * 65535.

`args.s_compr`

Indicates if the function removes silence. A value of 1 enables silence removal, 0 disables. Do not use this parameter during full-duplex recording and playback or if *args.asr_mode* is in use. Use *args.vad* in these cases.

`args.func`

Pointer to a user-supplied function called during speech recording. `NULL` disables this feature.

The *args.func* argument is called as `(*args.func)(lp,args.arg)`.

The *lp* variable contains the pointer to the line structure; *arg* contains the supplied user-defined argument.

Returns an integer:

0 Maintains speech recording.

1 Aborts speech recording.

`args.arg`

Argument to *args.func*. Can be `NULL`.

`args.vad`

Enables or disables Voice Activity Detection (VAD) and specifies the mode. Use this argument with full-duplex recording and playback, or when simplex recording and *args.asr_mode* is set to 1. Possible values are:

`VAD_OFF` 0

No VAD operation.

`VAD_ALL` 1

VAD reporting with all speech data supplied.

`VAD_DETECT` 2

VAD reporting with only detected speech data supplied.

`VAD_DETECT_AUTO` 3

VAD reporting with only detected speech data supplied and automatic playback suppression. Playback will no longer be audible, but the application must terminate the playback operation itself through one of the standard mechanisms for the playback function in use.

`VAD_THRESH` 4

VAD reporting with data that is above a threshold. Modify the threshold using *args.vad_thresh*.

`args.vad_thresh`

Specifies the change from the default detection sensitivity. Applicable to all VAD modes. Units are internally defined steps.

Range: –6 to 12

`args.vad_passthru`

When nonzero, enables unconditional pass through of the specified amount of data (in ms) when recording starts. This argument has no effect when *args.vad* is set to a value other than `VAD_OFF` or `VAD_ALL`.

`args.asr_mode`

When set to 1, speech recording takes place in an ASR-compatible mode, regardless of whether full-duplex speech operations are occurring.

`args.beep`

When nonzero, a beep will be played prior to record starting. If the channel is already performing playback, then the beep will not be played.

By default the beep tone will be played for 500 milliseconds at 500 Hz.

The behavior is controlled by the *record_beep_dur* and *record_beep_freq* parameters in the user-defined configuration file (*see Volume 6, Appendix A*).

`args.min_callback`

The maximum time (in milliseconds) that the Bfv API allows to elapse between calls to the user-supplied callback function (*args.func*). Zero (0) indicates that the calling of the callback function depends on the activity taking place on the channel.

Note: Setting this parameter to a nonzero value smaller than one second might have an adverse effect on the system CPU usage. The smaller the value, the greater the effect on CPU usage.

Output

Return value:

The number of bytes stored in *args.buf*. When this value is 0, speech recording has completed.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 supports only CODE_PCM_ULAW and CODE_PCM_ALAW coding formats, sampled at RATE_8000 samples per second with BITS_8 per sample. In addition, all VAD, ASR, and Summation parameters are also unsupported on virtual modules.

This function writes the received data to the user-allocated data buffer.

See *Table 25* through *Table 27*, starting on *page 569*, for information about valid speech settings. Not all combinations of *args.coding_fmt*, *args.rate*, and *args.bits_per_samp* are valid (see *Table 25, Voice Encoding Settings*, on *page 568*).

Recording stops at the first occurrence of one of the following:

- The value of *args.rate* is `RATE_STOP`
- Silence timeout
- Overall timeout
- The user-supplied integer function returns nonzero
- The line state changes to `IDLE`

After recording stops, the function stores the data remaining in the driver buffers into the supplied buffer and clears the driver buffers.

When recording is stopped because the value of rate is `RATE_STOP`, the driver buffer might still contain speech data. To receive the complete set of speech data, the application must call *BfvSpeechRecordData* repeatedly until the return value is 0.

The *args.timeout* and *args.silence_timeout* arguments only have effect when recording is first begun. Values for these arguments in subsequent calls to this function for the same recording are ignored. The *args.silence_timeout* argument has no effect during summation recording.

The application can enable Voice Activity Detection (VAD) for use with full-duplex recording and playback (often useful for ASR applications).

To enable AGC, the application uses the agc keyword in the user-defined configuration file.

The argument *args.func* can be NULL, or it can point to a user-supplied function that determines when to stop recording.

The user-supplied function is a flexible and powerful tool and can be used to:

- Detect the end of recording caused by an event. Events include:
  - Detection of a DTMF tone
  - Detection of a particular call progress result
  - An external trigger such as the keyboard
- Report information during recording.

The user-supplied function must not call any function that causes a delay, such as waiting for a DTMF tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing DTMF tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

If you set a nonzero value for *args.min_callback*, the Bfv API makes sure that it calls the *args.func* user-supplied callback function at least every *args.min_callback* milliseconds. If you set *args.min_callback* to zero, callback function calling depends on activity taking place on the channel.

During any of the following conditions, the user callback function might not be called regularly unless there is some activity destined for the application session. Alerts sent by *BfvLineAlert* (which can be cancelled by the application) can create such activity.

- When silence compression is enabled using *args.s_compr* and silence is detected.
- If VAD is enabled and no speech is detected.
- If speech data has been routed to an alternate timeslot using *BfvSpeechEchoCancelControl*.

If VAD is enabled, when the recording firmware determines that speech has started or stopped, use the LINE_VAD_STATE (lp) macro to determine the type of voice activity. This macro returns either VAD_SPEECH_DETECTED to indicate speech was detected or VAD_NO_SPEECH_DETECTED (default state) to indicate that speech

is not currently being detected. Your application would normally call this macro from within the user callback function. However, the macro can also be called from outside the user callback function. The `LINE_VAD_BYTES_PROCESSED (lp)` macro returns the number of bytes played at the time the last speech detection occurred.

After full-duplex operation begins or when *args.asr_mode* is in use, the application assumes that it stays in effect until both recording and playback have been terminated.

In full-duplex operation or when *args.asr_mode* is in use, the silence compression and AGC capabilities are not available. When performing full-duplex recording/playing, the record coding format value can only be `CODE_PCM_ULAW` or `CODE_PCM_ALAW`, but the play format can be any valid format.

See Also

BfvSpeechRecord, *BfvSpeechRecord*, *BfvSpeechRecordFile*, *BfvSpeechRecordWave*,
`LINE_HAS_CAP (lp, cap)`, `LINE_VAD_STATE (lp)`, `LINE_VAD_BYTES_PROCESSED (lp)`

Example

```
BTLINE *lp;
unsigned char buf[1024];
int n;
int my_user_func();
struct args_speech args;

/* Record & write to a file until user */
/* function indicates to stop          */
for (;;)
{
    BT_ZERO(args);
    args.buf = buf;
    args.size = sizeof(buf);
    args.coding_fmt = CODE_ADPCM;
    args.rate = RATE_8000;
    args.bits_per_samp = BITS_4;
    args.afe_rate = AFE_8000;
    args.data_fmt = SPCH_MSB;
    args.timeout = 0L;
    args.silence_timeout = 0L;
    args.s_compr = 0;
    args.func = my_user_func;
    args.arg = NULL;
    if ((n = BfvSpeechRecordData(lp, &args)) <= 0 ||
     args.res.status != BT_STATUS_OK)
     return(-1);
    write_file(buf, n);
}
```

BfvSpeechRecordFile

Purpose

Records raw speech data into the specified file using the specified speech parameters.

Syntax

```
void
BfvSpeechRecordFile          (lp, args)
   BTLINE                    *lp;
   struct args_speech        *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
unsigned char *buf;
unsigned size;
unsigned coding_fmt;
unsigned rate;
unsigned bits_per_samp;
unsigned afe_rate;
unsigned data_fmt;
long timeout;
long silence_timeout;
int s_compr;
int (*func)(BTLINE *lp, char *arg);
char *arg;
int vad;
int vad_thresh;
unsigned vad_passthrugh;
int asr_mode;
int beep;
unsigned min_callback;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fname

Name of the file in which to store the incoming data.

args.buf

Pointer to the user-allocated data buffer that holds the received data. The data buffer must be at least 1024 bytes in size.

args.size

The size, in bytes, of the user-allocated data buffer.

args.coding_fmt

Coding format of the data. Valid values are:

CODE_ADPCM OKI ADPCM	1
CODE_PCM_ULAW µ-Law	2
CODE_PCM_ALAW A-Law	3
CODE_G726 G.726	17
CODE_GSM_610 Microsoft GSM 6.10	14
CODE_LINEAR Linear	4

`args.rate`

The compressed sample rate for playback. The lower rates are more space efficient but produce poorer speech quality. If the value is `RATE_STOP`, speech playback is stopped. Valid values are:

`RATE_STOP`	
Stop playback	
`RATE_5300`	9
5,300 samples/sec	
`RATE_6000`	0
6,000 samples/sec	
`RATE_8000`	1
8,000 samples/sec	
`RATE_9600`	7
9,600 samples/sec	
`RATE_11000`	8
11,000 samples/sec	
`RATE_12300`	12
12,300 samples/sec	
`RATE_13000`	11
13,000 samples/sec	
`RATE_16000`	6
16,000 samples/sec	
`RATE_20000`	2
20,000 samples/sec	
`RATE_24000`	3
24,000 samples/sec	
`RATE_28000`	4
28,000 samples/sec	
`RATE_32000`	5
32,000 samples/sec	

`args.bits_per_samp`

Number of bits per sample. Valid values are:

`BITS_1` 0
: 1 bit/sample

`BITS_2` 7
: 2 bit/sample

`BITS_3` 1
: 3 bit/sample

`BITS_4` 2
: 4 bits/sample

`BITS_5` 8
: 5 bit/sample

`BITS_8` 3
: 8 bits/sample

`BITS_16` 5
: 16 bits/sample

`args.afe_rate`

AFE sample rate. Valid values are:

`AFE_8000` 0
: 8000 samples/sec

`args.data_fmt`

Data format specification. Value at present is:

`SPCH_MSB` 0
: MSB is first

`args.timeout`

If nonzero, recording terminates after no more than *args.timeout* milliseconds elapse.

`args.silence_timeout`

Stops recording when silence is detected for this amount of time (in milliseconds). Zero (0) indicates that recording continues unaffected by a period of silence. Maximum value is 5 * 65535.

`args.s_compr`

Indicates if the function removes silence. A value of 1 enables silence removal, 0 disables. Do not use this parameter during full-duplex recording and playback or if *args.asr_mode* is in use. Use *args.vad* in these cases.

`args.func`

Pointer to a user-supplied function called during speech recording. `NULL` disables this feature.

The *args.func* argument is called as `(*args.func)(lp,args.arg)`.

The *lp* variable contains the pointer to the line structure; *arg* contains the supplied user-defined argument.

Returns an integer:

0 Maintains speech recording.

1 Aborts speech recording.

`args.arg`

Argument to *args.func*. Can be `NULL`.

`args.vad`

Enables or disables Voice Activity Detection (VAD) and specifies the mode. Use this argument with full-duplex recording and playback, or when simplex recording and *args.asr_mode* is set to 1. Possible values are:

`VAD_OFF` 0

No VAD operation.

`VAD_ALL` 1

VAD reporting with all speech data supplied.

`VAD_DETECT` 2

VAD reporting with only detected speech data supplied.

`VAD_DETECT_AUTO` 3

VAD reporting with only detected speech data supplied and automatic playback suppression. Playback will no longer be audible, but the application must terminate the playback operation itself through one of the standard mechanisms for the playback function in use.

`VAD_THRESH` 4

VAD reporting with data that is above a threshold. Modify the threshold using *args.vad_thresh*.

`args.vad_thresh`

Specifies the change from the default detection sensitivity. Applicable to all VAD modes. Units are internally defined steps.

Range: –6 to 12

`args.vad_passthru`

When nonzero, enables unconditional pass through of the specified amount of data (in ms) when recording starts. This argument has no effect when *args.vad* is set to a value other than `VAD_OFF` or `VAD_ALL`.

`args.asr_mode`

When set to 1, speech recording takes place in an ASR-compatible mode, regardless of whether full-duplex speech operations are occurring.

`args.beep`

When nonzero, a beep will be played prior to record starting. If the channel is already performing playback, then the beep will not be played.

By default the beep tone will be played for 500 milliseconds at 500 Hz.

The behavior is controlled by the *record_beep_dur* and *record_beep_freq* parameters in the user-defined configuration file (*see Volume 6, Appendix A*).

`args.min_callback`

The maximum time (in milliseconds) that the Bfv API allows to elapse between calls to the user-supplied callback function (*args.func*). Zero (0) indicates that the calling of the callback function depends on the activity taking place on the channel.

Note: Setting this parameter to a nonzero value smaller than one second might have an adverse effect on the system CPU usage. The smaller the value, the greater the effect on CPU usage.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 supports only CODE_PCM_ULAW and CODE_PCM_ALAW coding formats, sampled at RATE_8000 samples per second with BITS_8 per sample. In addition, all VAD, ASR, and Summation parameters are also unsupported on virtual modules.

This function writes the received data to the user-specified file.

See *Table 25* through *Table 27*, starting on *page 569*, for information about valid speech settings. Not all combinations of *args.coding_fmt*, *args.rate*, and *args.bits_per_samp* are valid (see *Table 25, Voice Encoding Settings*, on *page 568*).

Recording stops at the first occurrence of one of the following:

- The value of *args.rate* is `RATE_STOP`
- Silence timeout
- Overall timeout
- The user-supplied integer function returns nonzero
- The line state changes to `IDLE`

After recording stops, the function stores the data remaining in the driver buffers into the supplied file and clears the driver buffers.

When recording is stopped because the value of rate is `RATE_STOP`, the driver buffer might still contain speech data. To receive the complete set of speech data, the application must call *BfvSpeechRecordFile* repeatedly until the return value is 0.

The *args.timeout* and *args.silence_timeout* arguments only have effect when recording is first begun. Values for these arguments in subsequent calls to this function for the same recording are ignored. The *args.silence_timeout* argument has no effect during summation recording.

The application can enable Voice Activity Detection (VAD) for use with full-duplex recording and playback (often useful for ASR applications).

To enable `AGC`, the application uses the `agc` keyword in the user-defined configuration file.

The argument *args.func* can be `NULL`, or it can point to a user-supplied function that determines when to stop recording.

The user-supplied function is a flexible and powerful tool and can be used to:

- Detect the end of recording caused by an event. Events include:
  - Detection of a `DTMF` tone
  - Detection of a particular call progress result
  - An external trigger such as the keyboard
- Report information during recording.

The user-supplied function must not call any function that causes a delay, such as waiting for a `DTMF` tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing `DTMF` tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

If you set a nonzero value for *args.min_callback*, the Bfv API makes sure that it calls the *args.func* user-supplied callback function at least every *args.min_callback* milliseconds. If you set *args.min_callback* to zero, callback function calling depends on activity taking place on the channel.

During any of the following conditions, the user callback function might not be called regularly unless there is some activity destined for the application session. Alerts sent by *BfvLineAlert* (which can be cancelled by the application) can create such activity.

- When silence compression is enabled using *args.s_compr* and silence is detected.
- If VAD is enabled and no speech is detected.
- If speech data has been routed to an alternate timeslot using *BfvSpeechEchoCancelControl*.

If VAD is enabled, when the recording firmware determines that speech has started or stopped, use the `LINE_VAD_STATE (lp)` macro to determine the type of voice activity. This macro returns either `VAD_SPEECH_DETECTED` to indicate speech was detected or `VAD_NO_SPEECH_DETECTED` (default state) to indicate that speech is not currently being detected. Your application would normally call this macro from within the user callback function. However, the macro can also be called from outside the user callback function. The `LINE_VAD_BYTES_PROCESSED (lp)` macro returns the number of bytes played at the time the last speech detection occurred.

After full-duplex operation begins or when *args.asr_mode* is in use, the application assumes that it stays in effect until both recording and playback have been terminated.

In full-duplex operation or when *args.asr_mode* is in use, the silence compression and AGC capabilities are not available. When performing full-duplex recording/playing, the record coding format value can only be `CODE_PCM_ULAW` or `CODE_PCM_ALAW`, but the play format can be any valid format.

See Also

BfvSpeechRecord, *BfvSpeechRecord*, *BfvSpeechRecordData*, *BfvSpeechRecordWave*, `LINE_HAS_CAP (lp, cap)`, `LINE_VAD_STATE (lp)`, `LINE_VAD_BYTES_PROCESSED (lp)`

Example

```
BTLINE *lp;
char *fname;
int my_user_func();
struct args_speech args;

BT_ZERO(args);
args.fname = fname;
args.coding_fmt = CODE_ADPCM;
args.rate = RATE_8000;
args.bits_per_samp = BITS_4;
args.afe_rate = AFE_8000;
args.data_fmt = SPCH_MSB;
args.timeout = 10000L;
args.silence_timeout = 0L;
args.s_compr = 0;
args.func = my_user_func;
args.arg = NULL;
BfvSpeechRecordFile(lp, &args);
```

BfvSpeechRecordWave

Purpose

Records speech into the specified wave (*.wav*) file using the specified speech parameters.

Syntax

```
void
BfvSpeechRecordWave          (lp, args)
   BTLINE                    *lp;
   struct args_speech        *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
unsigned char *buf;
unsigned size;
unsigned coding_fmt;
unsigned rate;
unsigned bits_per_samp;
unsigned afe_rate;
unsigned data_fmt;
long timeout;
long silence_timeout;
int s_compr;
int (*func)(BTLINE *lp, char *arg);
char *arg;
int vad;
int vad_thresh;
unsigned vad_passthrough;
int asr_mode;
int beep;
unsigned min_callback;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.fname

Name of wave file to record.

args.buf

Pointer to the user-allocated data buffer that holds the received data. The data buffer must be at least 1024 bytes in size.

args.size

The size, in bytes, of the user-allocated data buffer.

args.coding_fmt

Coding format of the data. Valid values are:

`CODE_PCM_ULAW`	2
μ-Law	
`CODE_PCM_ALAW`	3
A-Law	
`CODE_LINEAR`	4
Linear	

args.rate

The compressed sample rate for recording. Valid values are:

`RATE_8000`	1
8,000 samples/sec	
`RATE_11000`	8
11,025 samples/sec	

`args.bits_per_samp`

Number of bits per sample. Valid values are:

`BITS_8` 3

8 bits/sample

`BITS_16` 5

16 bits/sample

`args.afe_rate`

AFE sample rate. Valid values are:

`AFE_8000` 0

8000 samples/sec.

`args.data_fmt`

Data format specification. Value at present is:

`SPCH_MSB` 0

MSB is first

`args.timeout`

The total time (in milliseconds) to record. Zero (0) indicates that recording continues unaffected by a time limit.

`args.silence_timeout`

Stops recording when silence is detected for this amount of time (in milliseconds). Zero (0) indicates that recording continues unaffected by a period of silence. Maximum value is 5 * 65535.

`args.s_compr`

Indicates if the function removes silence. A value of 1 enables silence removal, 0 disables. Do not use this parameter during full-duplex recording and playback or if *args.asr_mode* is in use. Use *args.vad* in these cases.

`args.func`

Pointer to a user-supplied function called during speech recording. `NULL` disables this feature.

The *args.func* argument is called as `(*args.func)(lp,args.arg)`.

The *lp* variable contains the pointer to the line structure; *arg* contains the supplied user-defined argument.

Returns an integer:

0 Maintains speech recording.

1 Aborts speech recording.

`args.arg`

Argument to *args.func*. Can be `NULL`.

`args.vad`

Enables or disables Voice Activity Detection (VAD) and specifies the mode. Use this argument with full-duplex recording and playback, or when simplex recording and *args.asr_mode* is set to 1. Possible values are:

`VAD_OFF` 0

No VAD operation.

`VAD_ALL` 1

VAD reporting with all speech data supplied.

`VAD_DETECT` 2

VAD reporting with only detected speech data supplied.

`VAD_DETECT_AUTO` 3

VAD reporting with only detected speech data supplied and automatic playback suppression. Playback will no longer be audible, but the application must terminate the playback operation itself through one of the standard mechanisms for the playback function in use.

`VAD_THRESH` 4

VAD reporting with data that is above a threshold. Modify the threshold using *args.vad_thresh*.

`args.vad_thresh`

Specifies the change from the default detection sensitivity. Applicable to all VAD modes. Units are internally defined steps.

Range: –6 to 12

`args.vad_passthru`

When nonzero, enables unconditional pass through of the specified amount of data (in ms) when recording starts. This argument has no effect when *args.vad* is set to a value other than `VAD_OFF` or `VAD_ALL`.

`args.asr_mode`

When set to 1, speech recording takes place in an ASR-compatible mode, regardless of whether full-duplex speech operations are occurring.

`args.beep`

When nonzero, a beep will be played prior to record starting. If the channel is already performing playback, then the beep will not be played.

By default the beep tone will be played for 500 milliseconds at 500 Hz.

The behavior is controlled by the *record_beep_dur* and *record_beep_freq* parameters in the user-defined configuration file (*see Volume 6, Appendix A*).

`args.min_callback`

The maximum time (in milliseconds) that the Bfv API allows to elapse between calls to the user-supplied callback function (*args.func*). Zero (0) indicates that the calling of the callback function depends on the activity taking place on the channel.

Note: Setting this parameter to a nonzero value smaller than one second might have an adverse effect on the system CPU usage. The smaller the value, the greater the effect on CPU usage.

Output

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The SR140 supports only CODE_PCM_ULAW and CODE_PCM_ALAW coding formats, sampled at RATE_8000 samples per second with BITS_8 per sample. In addition, all VAD, ASR, and Summation parameters are also unsupported on virtual modules.

This function writes the received data into the specified *.wav* file.

See *Table 25* through *Table 27*, starting on *page 569*, for information about valid speech settings. Because some format or rate conversions might be done at host level, the possible formats and rates might not match the native capabilities of the firmware (see *Table 25, Voice Encoding Settings*, on *page 568*).

Recording stops at the first occurrence of one of the following:

- The value of *args.rate* is `RATE_STOP`
- Silence timeout
- Overall timeout
- The user-supplied integer function returns nonzero
- The line state changes to `IDLE`

After recording stops, the function stores the data remaining in the driver buffers into the supplied file and clears the driver buffers.

When recording is stopped because the value of rate is `RATE_STOP`, the driver buffer might still contain speech data. To receive the complete set of speech data, the application must call *BfvSpeechRecordWave* repeatedly until the return value is 0.

The *args.timeout* and *args.silence_timeout* arguments only have effect when recording is first begun. Values for these arguments in subsequent calls to this function for the same recording are ignored. The *args.silence_timeout* argument has no effect during summation recording.

The application can enable Voice Activity Detection (VAD) for use with full-duplex recording and playback (often useful for ASR applications).

To enable `AGC`, the application uses the `agc` keyword in the user-defined configuration file.

The argument *args.func* can be `NULL`, or it can point to a user-supplied function that determines when to stop recording.

The user-supplied function is a flexible and powerful tool and can be used to:

- Detect the end of recording caused by an event. Events include:
  - Detection of a DTMF tone
  - Detection of a particular call progress result
  - An external trigger such as the keyboard
- Report information during recording.

The user-supplied function must not call any function that causes a delay, such as waiting for a DTMF tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing DTMF tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

If you set a nonzero value for *args.min_callback*, the Bfv API makes sure that it calls the *args.func* user-supplied callback function at least every *args.min_callback* milliseconds. If you set *args.min_callback* to zero, callback function calling depends on activity taking place on the channel.

During any of the following conditions, the user callback function might not be called regularly unless there is some activity destined for the application session. Alerts sent by *BfvLineAlert* (which can be cancelled by the application) can create such activity.

- When silence compression is enabled using *args.s_compr* and silence is detected.
- If VAD is enabled and no speech is detected.
- If speech data has been routed to an alternate timeslot using *BfvSpeechEchoCancelControl*.

If VAD is enabled, when the recording firmware determines that speech has started or stopped, use the LINE_VAD_STATE (lp) macro to determine the type of voice activity. This macro returns either VAD_SPEECH_DETECTED to indicate speech was detected or VAD_NO_SPEECH_DETECTED (default state) to indicate that speech is not currently being detected. Your application would normally call this macro from within the user callback function. However, the macro can also be called from outside the user callback function. The LINE_VAD_BYTES_PROCESSED (lp) macro returns the number of bytes played at the time the last speech detection occurred.

After full-duplex operation begins or when *args.asr_mode* is in use, the application assumes that it stays in effect until both recording and playback have been terminated.

In full-duplex operation or when *args.asr_mode* is in use, the silence compression and AGC capabilities are not available. When performing full-duplex recording/playing, the record coding format value can only be `CODE_PCM_ULAW` or `CODE_PCM_ALAW`, but the play format can be any valid format.

See Also

BfvSpeechRecord, *BfvSpeechRecord*, *BfvSpeechRecordData*, *BfvSpeechRecordFile*,
`LINE_HAS_CAP (lp, cap)`, `LINE_VAD_STATE (lp)`, `LINE_VAD_BYTES_PROCESSED (lp)`

Example

```
BTLINE *lp;
char *fname;
int my_user_func();
struct args_speech args;

BT_ZERO(args);
args.fname = fname;
args.coding_fmt = CODE_PCM_ULAW;
args.rate = RATE_8000;
args.bits_per_samp = BITS_8;
args.afe_rate = AFE_8000;
args.data_fmt = SPCH_MSB;
args.timeout = 10000L;
args.silence_timeout = 0L;
args.s_compr = 0;
args.func = my_user_func;
args.arg = NULL;
BfvSpeechRecordWave(lp, &args);
```

Macros

LINE_HAS_CAP (*lp*, *cap*)

Determines if a channel has a particular capability. Returns nonzero if *lp* has capability *cap*; returns 0 if *lp* does not have capability *cap*. The *caps.h* header file contains definitions of the capabilities. You should use this macro instead of examining the `LINE_TYPE` value.

LINE_VAD_STATE *(lp)*

When recording with Voice Activity Detection (VAD) enabled, this macro returns either `VAD_SPEECH_DETECTED` to indicate that speech was detected or `VAD_NO_SPEECH` (default state) to indicate that speech is not currently being detected. Your application would normally call this macro from within the user callback function. However, it can also be called from outside the user callback function.

LINE_VAD_BYTES_PROCESSED *(lp)*

When recording with Voice Activity Detection (VAD) enabled, this macro returns the number of bytes played at the time the last speech detection occurred.

16 - Infopkt File Functions

This chapter describes the functions used to process infopkt files.

The chapter provides syntax and code examples for the functions used to process infopkt files. Functions used to manage speech playback and record using infopkts or fax transmission and reception using infopkts are described elsewhere.

InfoPkt Function Summary

Table 1 provides a high-level description of each Infopkt function. Detailed descriptions of each function begin on *page 601*.

Table 1. Infopkt Function Summary

Function	Purpose	Page
BfvInfopktClose	Closes the current infopkt file and frees all associated structure memory.	*601*
BfvInfopktFseek	Searches to a specified offset in a file relative to a specified origin.	*603*
BfvInfopktFtell	Gives the pointer position within an infopkt stream file.	*605*
BfvInfopktGet	Reads one infopkt from the stream. Can follow indirect infopkts if specified.	*607*
BfvInfopktOpen	Opens an infopkt stream-formatted disk file.	*609*
BfvInfopktOpenMem	Opens an infopkt stream associated with a user-supplied buffer instead of a file.	*611*
BfvInfopktPut	Writes one infopkt to the infopkt stream file.	*615*
BfvInfopktUnget	Replaces the last infopkt in the infopkt stream so it is available for the next request.	*617*
BfvInfopktUser	Sets up a user-supplied function to handle user-defined infopkts.	*619*
BfvPromptClose	Closes the current prompt file and frees its associated memory.	*622*
BfvPromptOpen	Opens a prompt file, a specialized infopkt file.	*624*

BfvInfopktClose

Purpose

Closes the current file and frees all associated structure memory.

Syntax

```
int
BfvInfopktClose              (args)
   struct args_infopkt       *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *ips;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.ips

Pointer to the infopkt stream.

Output

Return value:

0	The current file was closed successfully.
nonzero	An error occurred in closing the file.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function retains the same resource-sharing restrictions as the *fopen* call.

Use the *BfvInfopktOpenMem* function to open memory-based infopkt streams.

If the infopkt stream was opened using the *args.use_open_file* option, calling this function does not close the originally opened file.

See Also

BfvInfopktOpen

Example

```
struct infopkt_stream *ips;
struct args_infopkt args;

BT_ZERO(args);
args.ips = ips;
BfvInfopktClose(&args);
```

BfvInfopktFseek

Purpose

Searches to a specified offset in a file relative to the specified origin.

Syntax

```
int
BfvInfopktFseek              (args)
   struct args_infopkt      *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *ips;
long offset;
int origin;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.ips

Pointer to the infopkt stream.

args.offset

The offset in the file from the specified origin.

args.origin

0 Beginning of file.

1 Current location in file.

2 End of file.

Output

Return value:

0	The current file was closed successfully.
nonzero	An error occurred in closing the file.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function is not useful while the application is processing nested infopkt files.

In some compilation environments, the application can use the symbolic constants `SEEK_SET`, `SEEK_CUR`, and `SEEK_END` for the origin values. See your *fseek* documentation for more details.

See Also

BfvInfopktFtell

Example

```
struct infopkt_stream *ips;
struct args_infopkt args;

BT_ZERO(args);
args.ips = ips;
args.offset = 0L;
args.origin = 0;
err = BfvInfopktFseek(&args);
```

BfvInfopktFtell

Purpose

Retrieves the position of the pointer within the specified infopkt stream file.

Syntax

```
long
BfvInfopktFtell              (args)
   struct args_infopkt      *args;
```

The structure contains the following fields.

Input Fields

struct infopkt_stream **ips;*

Output Fields

RES *res;*

Input

args

Pointer to an argument structure containing input and output fields.

args.ips

Pointer to the infopkt stream.

Output

Return value:

The position of the pointer as the number of bytes from the beginning of the infopkt stream file.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function is not useful while the application is processing nested infopkt files.

See Also

BfvInfopktFseek

Example

```
struct infopkt_stream *ips;
long position;
struct args_infopkt args;

BT_ZERO(args);
args.ips = ips;
position = BfvInfopktFtell(&args);
```

BfvInfopktGet

Purpose

Gets infopkts from an infopkt stream and can follow indirect infopkts if specified.

Syntax

```
struct infopkt              *
BfvInfopktGet               (args)
   struct args_infopkt      *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *ips;
enum indir_mode i_mode;
```

Output Fields

```
RES res;
```

Modified Fields

offset, *origin*

Input

args

Pointer to an argument structure containing input and output fields.

args.ips

Pointer to the infopkt stream.

args.i_mode

INDIR_MODE_FOLLOW

Indirect infopkts are followed.

INDIR_MODE_RETURN

Indirect infopkts are not followed; they are returned.

INDIR_MODE_FOLLOW_NOUSER

User-defined infopkts are ignored.

Output

Return value:

Infopkt pointer. An infopkt read from the stream *args.ips*.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

Reads one infopkt from the stream and can follow indirect infopkts if specified.

Returned data is allocated in the *args.ips* structure and is overwritten by the results of the next *BfvInfopktGet* function called on the same stream.

Indirect infopkts are followed if that parameter is selected.

Several infopkt streams can operate simultaneously without interfering with each other.

User-defined infopkts can be processed with the *BfvInfopktUser* function when `INDIR_MODE_FOLLOW_NOUSER` mode is used.

See *Accessing an Infopkt Stream From an Application* in *Chapter 2* of the *Dialogic® Brooktrout® Fax Products SDK Developer Guide* for more information.

See Also

BfvInfopktUnget

Example

```
struct infopkt_stream *ips;
struct infopkt *ip;
struct args_infopkt args;

BT_ZERO(args);
args.ips = ips;
args.i_mode = INDIR_MODE_FOLLOW;
if ((ip = BfvInfopktGet(&args) == NULL)
    printf("Unable to get infopkt\n");
```

BfvInfopktOpen

Purpose

Opens an infopkt stream-formatted disk file.

Syntax

```
struct infopkt_stream      *
BfvInfopktOpen             (args)
   struct args_infopkt     *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
char *fmode;
int use_open_file;
FILE *fp;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.fname

Complete filename, including the path, of the disk file.

args.fmode

Sets the mode to read, write, or append.

"r" Read

"w" Write

"a" Append

args.use_open_file

When set to 1, the function creates an infopkt stream using the `FILE *` file pointer supplied in *args.fp*.

args.fp

`FILE *` file pointer for use with *use_open_file*.

Output

Return value:

`infopkt_stream` pointer	=	File successfully opened.
`NULL`	=	Error occurred.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function retains the same resource-sharing restrictions as the *fopen* call. Applications use the *BfvInfopktOpenMem* function to open memory-based infopkt streams.

See Also

BfvInfopktClose, *BfvInfopktOpenMem*, *BfvInfopktGet*, *BfvInfopktPut*

Example

```
struct infopkt_stream *ips;
struct args_infopkt args;

BT_ZERO(args);
args.fname = "myinfopkt";
args.fmode = "r";
if ((ips = BfvInfopktOpen(&args)) == NULL)
    fprintf(stderr, "Can't open infopkt\n");
```

BfvInfopktOpenMem

Purpose

Opens an infopkt stream associated with a user-supplied memory buffer instead of a file.

Syntax

```
struct infopkt_stream      *
BfvInfopktOpenMem          (args)
   struct args_infopkt     *args;
```

The structure contains the following fields.

Input Fields

```
unsigned char *buf;
long buf_size;
int limited_read;
long valid_data;
void (*buf_func)  (struct infopkt_stream *ips,
                   long old_offset,
                   long new_offset,
                   unsigned char **buf,
                   long *bufsize, char *arg);
char *arg;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.buf

Pointer to a user-supplied buffer for writing or reading infopkt data.

args.buf_size

The size, in bytes, of *args.buf*.

`args.limited_read`

If 0, then *BfvInfopktGet* can read the entire *args.buf_size* bytes. If 1, then *BfvInfopktGet* can read only *args.valid_data* bytes or the furthest data written to the buffer using *BfvInfopktPut*, whichever is larger.

`args.valid_data`

If *args.limited_read* is set to 1, then this specifies a limit for *BfvInfopktGet* as described under *args.limited_read*. The value must be <= *args.buf_size* unless a non-`NULL` value is supplied for *args.buf_func*.

`args.buf_func`

Pointer to a user-supplied callback function that the Bfv API calls if a call to either *BfvInfopktGet* or *BfvInfopktPut* would cause an access outside of the valid data in the current buffer. `NULL` disables this feature.

The *args.buf_func* argument is called as:

```
(*args.buf_func)(ips,old_offset,new_offset,buf,
 bufsize,args.arg);
```

The *old_offset* variable contains the current offset of the buffer within the file. The *new_offset* variable contains the required new offset. This value could be either greater than or less than the old offset. The *buf* variable is a char ** pointer pointing to a variable containing the current buffer value. The *bufsize* variable is a long * pointer pointing to a variable containing the current buffer length. The application modifies the buffer pointer and buffer length variables to contain the new buffer and length values. If the requested offset is not one the application wishes to service, it should set the length to 0. The *args.arg* argument contains the supplied user-defined argument.

`args.arg`

Argument for *args.buf_func*. Can be `NULL`.

Output

Return value:

`infopkt_stream` pointer	File successfully opened.
`NULL`	Error occurred.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

A memory-based infopkt stream does not require specifying an explicit read or write mode. You can use it for either purpose.

Any function that can use a file-based infopkt stream can use the infopkt stream that the *BfvInfopktOpenMem* function opens.

Use the *BfvInfopktOpen* function to open file-based infopkt streams.

When *BfvInfopktOpenMem* is called with *args.limited_read* set to 0 and *args.buf_func* set to `NULL`, any write attempt using *BfvInfopktPut* past *args.buf_size* gets an error; any read attempt using *BfvInfopktGet* past *args.buf_size* gets an error, and all *buf_size* bytes are considered valid data for read.

If *args.limited_read* is set to 1, only *args.valid_data* bytes of the *args.buf_size* total bytes are considered valid, and read attempts are limited to this amount. If writes are performed which go beyond *args.valid_data*, the valid size is increased.

If an *args.buf_func* is supplied, then this action allows replacement of the current buffer with a new buffer. The buffer can be considered representing a piece of a larger file, starting at some offset in the file and extending for some length. Initially the offset in the file is 0, and the length is *args.buf_size*. When *args.buf_func* is called, this application can change this offset.

By using the *args.limited_read*, *args.valid_data*, and *args.buf_func* input fields, memory-based infopkts can be used for both read and write in a powerful, consistent fashion. It is not necessary to have a buffer containing the entire set of data in memory at once.

See Also

BfvInfopktClose, *BfvInfopktOpen*

Example

```
struct infopkt_stream *ips;
unsigned char buffer[10000];
int buf_size;
FILE *fp;
struct args_infopkt args;
fp = fopen("test.pkt", "r");   /* or "rb" */
if (fp == NULL)
{
    fprintf(stderr,"Can't open file\n");
    exit(1);
}
buf_size = fread(buffer, 1, sizeof(buffer), fp);
fclose(fp);
BT_ZERO(args);
args.buf = buffer;
args.buf_size = buf_size;
if ((ips = BfvInfopktOpenMem(&args)) == NULL)
    fprintf(stderr, "Can't open mem-infopkt\n");
```

BfvInfopktPut

Purpose

Writes one infopkt to the infopkt stream.

Syntax

```
int
BfvInfopktPut                (args)
   struct args_infopkt       *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *ips;
struct infopkt *ip;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.ips

Pointer to the infopkt stream.

args.ip

Pointer to the infopkt.

Output

Return value:

0	Last infopkt successfully replaced in the infopkt stream.
<0	Failure to replace the last infopkt into the infopkt stream.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Only one infopkt can be returned to the infopkt stream. To back up up more than one infopkt, you must write your own application-level front end to these functions.

Example

```
struct infopkt_stream *ips;
struct infopkt *ip;
struct args_infopkt args;

BT_ZERO(args);
args.ips = ips;
args.ip = ip;
if (BfvInfopktPut(&args) < 0)
    printf("BfvInfopktPut failed\n");
```

BfvInfopktUnget

Purpose

Replaces the last infopkt in the infopkt stream so it is available for the next *BfvInfopktGet* call.

Syntax

```
int
BfvInfopktUnget              (args)
   struct args_infopkt       *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *ips;
struct infopkt *ip;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.ips

Pointer to the infopkt stream.

args.ip

Pointer to the infopkt.

Output

Return value:

0	Last infopkt successfully replaced in the infopkt stream.
<0	Failure to replace the last infopkt into the infopkt stream.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Only one infopkt can be returned to the infopkt stream. To back up up more than one infopkt, you must write your own application-level front end to these functions.

See Also

BfvInfopktGet

Example

```
struct infopkt_stream *ips;
struct infopkt *ip
struct args_infopkt args_infopkt;

BT_ZERO(args);
args.ips = ips;
args.ip = ip;
if (BfvInfopktUnget(&args) < 0)
    printf("BfvInfopktUnget failed\n");
```

BfvInfopktUser

Purpose

Sets up a user-supplied function to handle user-defined infopkts.

Syntax

```
void
BfvInfopktUser              (args)
   struct args_infopkt      *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *ips;
void (*user_func) (struct infopkt *ip,
                   struct infopkt_stream *ips,
                   char *arg);
char *arg;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.ips

Pointer to the infopkt stream.

args.user_func

Pointer to a user-supplied function called when a user type infopkt is encountered. This function does not return a value.

The user-supplied function is called as:

```
(*args.user_func) (ip, args.ips, args.arg)
```

where *ip* is the user-defined infopkt being read, *args.ips* is the infopkt stream, and *args.arg* is the user-defined argument.

args.arg

Argument for *args.user_func*; can be NULL.

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function sets up the state of the infopkt stream *args.ips* so that the application calls the user-supplied function *args.user_func* when the application calls *BfvInfopktGet* function with its *args.indir_mode* argument set to `INDIR_MODE_FOLLOW_NOUSER` and then encounters a user-defined infopkt in the infopkt stream.

Do not use the user-supplied function to close the infopkt stream *ips*. Since *args.user_func* can be called during fax transmission or speech playing, do not call other fax or speech functions from within it.

The user-supplied function must not call any function that causes a delay, such as waiting for a `DTMF` tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing `DTMF` tone detection must enable the detection before beginning the operation that uses the user-supplied function, such as speech playback.

See Also

BfvInfopktGet

Example

```
BTLINE *lp;
struct infopkt_stream *ips;
struct args_infopkt args_infopkt;
struct args_fax args_fax;
void user_func();

...
BT_ZERO(args_infopkt);
args_infopkt.fname = "test.pkt";
args_infopkt.fmode = "r";
ips = BfvInfopktOpen(&args_infopkt);
args_infopkt.ips = ips;
args_infopkt.user_func = user_func;
args_infopkt.arg = NULL;
BfvInfopktUser(&args_infopkt);
BT_ZERO(args_fax);
args_fax.s_ips = ips;
args_fax.local_id = "local_id";
BfvFaxSend(lp,&args_fax);
...

user_func(ip, ips, arg)
struct infopkt *ip;
struct infopkt_stream *ips;
char *arg;
{
...
```

BfvPromptClose

Purpose

Closes the current prompt file and frees all associated memory.

Syntax

```
int
BfvPromptClose                (args)
   struct args_infopkt       *args;
```

The structure contains the following fields.

Input Fields

PROMPT_MAP *prompt_map;

Output Fields

RES *res;*

Modified Fields

ips

Input

args

Pointer to an argument structure containing input and output fields.

args.prompt_map

PROMPT_MAP pointer for the current file.

Output

Return value:

0	The current prompt file was closed successfully.
nonzero	An error occurred in closing the current prompt file.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

You cannot reuse the pointer after the function returns a closing error.

See Also

BfvPromptOpen

Example

```
BTLINE *lp;
struct args_infopkt args_info;
PROMPT_MAP *prompt_map;
...

BT_ZERO (args_info);
args_info.name = “prompt.ips”;
args_info.fmode = “r”;
prompt_map = BfvPromptOpen (&args_info);

args_info.prompt_map = prompt_map;
BfvPromptClose (&args_tel);
```

BfvPromptOpen

Purpose

Opens the specified prompt file, which is a specialized infopkt file.

Syntax

```
PROMPT_MAP *
BfvPromptOpen                (args)
   struct args_infopkt       *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
char *fmode;
int use_open_file;
struct infopkt_stream *ips;
```

Output Fields

```
RES res;
```

Modified Fields

offset, *origin*, *ips*, *i_mode*

Input

args

Pointer to an argument structure containing input and output fields.

args.fname

The name of the prompt file.

args.fmode

Sets the mode, as with *fopen* .

"r" Read.

"r+" Read/write.

This input field must be supplied regardless of the *args.use_open_file* setting.

args.use_open_file

When set to 1, uses an opened infopkt stream supplied in *args.ips*.

args.ips

If *args.use_open_file* is set to 1, supplies an opened infopkt stream.

Output

Return value:

Returns a PROMPT_MAP pointer if it successfully opens the specified prompt file. Returns NULL if an error occurs when opening the specified prompt file.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function can also be used with an opened infopkt stream, possibly returned by *BfvInfopktOpenMem*.

The function retains the same resource-sharing restrictions as the *fopen* call.

You cannot use this function to create a new prompt file.

If the application calls this function with an opened infopkt stream, it must use *BfvPromptClose*, not *BfvInfopktClose*, to close the infopkt stream.

See Also

BfvPromptClose

Example

```
BTLINE *lp;
struct args_infopkt args_info;
PROMPT_MAP *prompt_map;
...

BT_ZERO (args_info);
args_info.name = "prompt.ips";
args_info.fmode = "r";
prompt_map = BfvPromptOpen (&args_info);
```

Volume 4 - Fax Processing

About this Volume

- *Volume 4, Fax Processing,* provides information about the following Bfv API components:
  - Fax functions and macros
  - TIFF-F files functions and macros

18 - Fax Overview

This chapter provides an overview of the fax functionality, functions and file formats.

The Bfv API contains functions that provide easy, flexible control of inbound and outbound fax calls. It also provides easy access to the T.30 facsimile protocol and eliminates the need to understand every detail of the protocol's implementation.

High-level functions simplify the process of transmitting and receiving facsimiles. For example, the high-level function *BfvFaxSend* is constructed of these mid-level and low-level functions:

- *BfvFaxBeginSend*
- *BfvFaxEndOfDocument*
- *BfvFaxGetRemoteInfo*
- *BfvFaxSendPage*
- *BfvFaxSetLocalId*
- *BfvFaxWaitForTraining*

Mid-level functions provide more flexibility and control than the corresponding high-level functions, but they require more knowledge of and attention to the basic steps involved in sending and receiving facsimiles. Low-level functions provide the greatest flexibility and control, but they require extensive knowledge of and attention to the basic steps involved.

Since both the high-level and mid-level functions use infopkt files exclusively, the distinction between them is measured in the flexibility and control they provide. The low-level functions, however, use raw data files exclusively. Combining the high-, mid-, and low-level functions within the same application program is valid and useful.

TIFF-F fax functions are used exclusively to send and receive TIFF-F files, rather than formatted infopkt stream files, see *Chapter , TIFF-F Files Functions on page 781*. The remaining fax functions are used with raw files; some are also used with raw data files, TIFF-F files, and/or DCX files, see *Chapter , Fax Functions on page 630*. A set of utilities is also provided that let you manipulate raw G3 fax files (rather than raw files) from the command line.

19 - Fax Functions

This chapter describes the high and mid-level functions to send and receive TIFF, Raw, DCX or Infopkt formatted images.

Each of the following fax functions may apply only to specific types of files. The functions are marked to indicate a type as follows:

- Infopkt-formatted files [I]
- Raw data files [R]
- TIFF/F files [T]
- DCX files [D]

If the file type does not matter, no type is designated.

Fax Function Summary

Table 2 provides a brief summary of these functions.

Table 2. Fax Function Summary

Function	Purpose	Page
BfvDataFSK	Fills an FSK buffer with FSK data for debugging aid.	*634*
BfvFaxAbort	Stops a fax transmission or reception cleanly when possible.	*638*
BfvFaxBegin	Initiates fax transmit or receive for infopkt streams. Handles all variations of polling. [I]	*640*
BfvFaxBeginRaw	Initiates fax transmit or receive for raw fax data. Handles all variations of polling. [R,D]	*647*
BfvFaxBeginReceive	Sets parameters and instructs the channel to receive.	*654*
BfvFaxBeginSend	Begins fax transmission using an infopkt stream. [I]	*659*
BfvFaxBeginSendRaw	Initiates fax transmit for raw fax data. [R,D]	*662*
BfvFaxBeginSendTiff	Begins transmission using a TIFF-F file. [T]	*667*
BfvFaxBeginTiff	Initiates fax transmit or receive for TIFF-F files. Handles all variations of polling. [T]	*670*
BfvFaxDownloadFont	Loads a .fz8 font file to the channel as the specified font.	*675*
BfvFaxDownloadFontData	Downloads a supplied font from the buffer to the channel as the specified numbered font.	*679*
BfvFaxEndOfDocument	Transmits end-of-page with no more pages to follow.	*682*
BfvFaxEndReception	Waits for completion of the T.30 confirmation handshaking sequence. Called after receipt of last page.	*684*
BfvFaxGetRemoteInfo	Waits for and reports ID, DIS/DCS and NSF/NSS data.	*686*
BfvFaxHeader	Sets up headers or footers on all subsequent pages in a fax transmission.	*688*
BfvFaxNextPage	Transmits an end-of-page and new page setup, if appropriate, for use with infopkt streams. [I]	*692*

Table 2. Fax Function Summary (Continued)

Function	Purpose	Page
BfvFaxNextPageDCX	Transmits an end-of-page and new page setup, if appropriate, for use with DCX pages. [D]	695
BfvFaxNextPageRaw	Transmits an end-of-page and new page setup, if appropriate, for noninfopkt-formatted fax data. [R]	698
BfvFaxNextPageTiff	Transmits an end-of-page and new page setup, if appropriate, for use with TIFF-F files. [T]	701
BfvFaxPageParams	Sets the page parameters for subsequent pages of data.	704
BfvFaxPoll	Sends and/or receives faxes using infopkt streams. A high-level function. [I]	706
BfvFaxRcvPageDCX	Receives a fax page to a DCX file. [D]	711
BfvFaxRcvPageTiff	Receives a fax page to a TIFF-F file. [T]	713
BfvFaxReceive	Receives faxes using infopkt streams. A high-level function. [I]	716
BfvFaxReceiveData	Receives raw fax data into a user-supplied buffer. [R]	721
BfvFaxReceiveFile	Receives a raw fax page to a file. [R]	725
BfvFaxReceivePage	Receives a fax page to an infopkt stream. [I]	728
BfvFaxReceivePages	Receives multiple pages of fax data to an infopkt stream. [I]	730
BfvFaxSend	Sends faxes using infopkt streams. A high-level function. [I]	732
BfvFaxSendData	Sends raw fax data from a user-supplied buffer. [R]	736
BfvFaxSendFile	Sends a noninfopkt-formatted fax page from a file. [R]	738
BfvFaxSendPage	Sends an entire page from the infopkt stream to the driver buffer. Looks for an EOF or new page type infopkt before returning. [I]	741
BfvFaxSendPageDCX	Sends a fax page from a DCX file. [D]	744
BfvFaxSendPageTiff	Sends a fax page from a TIFF-F file. [T]	746
BfvFaxSetLocalId	Sets the local ID to a specified string.	748

Table 2. Fax Function Summary (Continued)

Function	Purpose	Page
BfvFaxSetNSF	Sets up NSF, NSC, and NSS messages for transmission to the remote host.	*750*
BfvFaxSetReceiveFmt	Sets the format of the received data. [I,R,T]	*753*
BfvFaxSetSubPwdSep	Sets up a SUB, PWD, or SEP FSK message to send to the remote host.	*756*
BfvFaxStripParams	Separates different strips of data and sets the strip parameters. [R,T,D]	*759*
BfvFaxT30Holdup	Causes the channel to wait during T.30 negotiations and calls a user-supplied function.	*763*
BfvFaxT30Params	Sets the T.30 parameters for transmission.	*769*
BfvFaxT4TimerParams	Obtains the T4 duration, the T4 attempt, the current T4 timer value and the T4 timer expiration, or sets a new T4 timer value.	*773*
BfvFaxWaitForTraining	Reports when training is complete or `turn_around` is indicated.	*776*

BfvDataFSK

Purpose

Fills a user-allocated buffer with all the FSK data that the line has sent and/or received since the last *BfvDataFSK* call.

Syntax

```
int
BfvDataFSK                  (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
unsigned char *buf;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.buf

Pointer to the beginning of the user-allocated buffer. This buffer must be allocated at least 1K bytes.

Output

Return value:

1	No new FSK data is available. Ignore the contents of the user buffer.
0	The contents of the FSK buffer were successfully transferred into the user buffer.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

FSK (*F*requency *S*hift *K*ey) data consists of commands and data that are issued during the T.30 protocol handshaking procedure and between pages of data transmission. Retrieving and reviewing FSK data after each function call facilitates debugging during the T.30 procedure.

To preserve chronological order, the contents of the FSK data buffer are discarded when wrap-around occurs. Because the FSK buffer is a 1K ring buffer, 1K blocks of FSK data can be lost when wrap-around occurs on unread data. No indication is passed to the user when the contents of the FSK buffer are discarded.

FSK data is encoded in the buffer as a sequence of elements in the form: *NSx*, where N specifies the number of bytes of FSK data that follow after the S byte, S identifies the following FSK data as either transmitted (1) or received (0), and x represents the bytes of actual FSK data (which includes an FSK command followed by its associated data). The N and S fields are each one byte, and the x field is limited to a maximum of 255 bytes. A zero in N indicates the end of the FSK data.

After each call, the FSK buffer is reset, and old data transferred into the user buffer cannot be recaptured after a *BfvDataFSK* command is issued.

Using the *BfvDataFSK* function in a loop causes an undesirable `busy-wait` and prevents updating of FSK values. To avoid this problem, use *BfvRcvProcessPkt*.

FSK Signals

For FSK signal definitions, see header file *fsk.h*.

CFR	Confirmation to receive.
CIG	Calling subscriber identification.
CRP	Command repeat.
CSI	Called subscriber identification.
CTC	Continue to correct.
CTR	Response to CTC.
DCN	Disconnect.

DCS	Digital command signal.
DIS	Digital identification signal.
DTC	Digital transmit command.
EOM	End of message.
EOP	End of procedures.
EOR	End of retransmission.
ERR	Response to EOR.
FTT	Failure to train.
MCF	Message confirmation.
MPS	Multipage signal.
NSC	Nonstandard facilities command.
NSF	Nonstandard facilities.
NSS	Nonstandard facilities setup.
PIN	Procedure interrupt negative.
PIP	Procedure interrupt positive.
PPR	Partial page request.
PPS	Partial page signal.
PRI-EOM	Procedure interrupt—End of message.
PRI-EOP	Procedure interrupt—End of procedures.
PRI-MPS	Procedure interrupt—Multipage signal.
PWD	Password.
RNR	Receive not ready.
RR	Receive ready.
RTN	Retrain negative.
RTP	Retrain positive.
SEP	Selective polling.
SUB	Subaddress.
TSI	Transmitting subscriber identification.

Example

```
void debug_fsk_cp_calls(lp)
BTLINE *lp;
{
    FILE *fp;
    unsigned char ubuf[1024];
    unsigned char *print;
    unsigned char count;
    int i;
    static char *dirstr[] = {"received", "sent"};
    char temp[20];
    struct args_fax args;

    sprintf(temp, "stats.%02d",LINE_UNIT_NUM(lp));
    fp = fopen(temp,"a");
    BT_ZERO(args);
    args.buf = ubuf;
    if (BfvDataFSK(lp,&args) == 0)
    {
        print = ubuf;
        while ((count = *print++) != 0)
        {
            fprintf(fp,"Count = %d(%s)\n",
                count, dirstr[*print++]);
            for (i=0; i < count; i++)
                fprintf(fp, " %x", *print++);
            fprintf(fp, "\n");
        }
    }
    else
        fprintf(fp, "FSK BUFFER empty error\n");
    fclose (fp);
}
```

BfvFaxAbort

Purpose

Stops a fax transmission or reception cleanly when possible.

Syntax

```
void
BfvFaxAbort                 (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Output Fields

```
RES res;
```

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Call this function only between Bfv API function calls or from within an interrupt function set up using the `LINE_SET_INCOMING_CMD_FUNC` macro. Do not call it from another thread.

This procedure usually includes sending a DCN to the remote fax machine.

After calling this function and until it finishes, which takes a few seconds, the application must attempt to continue from the point of interruption.

If the application cannot continue, it must call the *BfvFaxEndOfDocument* function on transmit or the *BfvFaxEndReception* function on receive.

Example

```
main()
{
    BTLINE *lp;
    struct infopkt_stream *ips;
    struct args_fax args;
    ...
    BT_ZERO(args);
    args.s_ips = ips;
    BfvFaxBeginSend(lp, &args);
    ...
    BfvFaxAbort(lp, &args);
    ...
    BfvFaxEndOfDocument(lp, &args);
}
```

BfvFaxBegin

Purpose

Initiates the fax process in originate or answer mode depending on whether the channel dialed a number or answered a call.

Syntax

```
void
BfvFaxBegin                    (lp, args)
   BTLINE                      *lp;
   struct args_fax             *args;
```

The structure contains the following fields.

Input Fields

```
int oa_flag;
struct infopkt_stream *s_ips;
struct infopkt_stream *r_ips;
int xmit_mode;
int rcv_mode;
int specify_tz_offset;
int tz_offset;
unsigned fallback_rtp_reinvite;
int ecm_override;
int override_restrict_res;
int restrict_res;
int override_eff_pt_caps;
int eff_pt_caps;
int override_max_width;
int max_width;
int v34_override;
int override_line_compression;
int line_compression;
```

Output Fields

```
RES res;
```

Modified Fields

can_send, can_receive, resolution, width, eff_page_type

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.oa_flag

Indicates whether the channel dialed a number or answered a call.

BT_ORIGINATE Channel dialed a number.

BT_ANSWER Channel answered a call.

args.s_ips

Pointer to the infopkt stream ready for transmission.

NULL = Sending is disabled.

args.r_ips

Pointer to the infopkt stream ready for fax data reception.

NULL = Receiving is disabled.

args.xmit_mode

Specifies the fax transmit mode. Valid values are:

XMIT_MODE_AUTO Auto transmit mode; standard, default.	0
XMIT_MODE_MANUAL Manual transmit mode; same as AUTO, but has timing differences when retransmitting FSK signals.	1

args.rcv_mode

Specifies the fax receive mode. Valid values are:

RCV_MODE_AUTO Auto receive mode; standard, default.	0
RCV_MODE_MANUAL Manual receive mode; same as AUTO, but no initial CED is played and startup delays are reduced.	1

args.specify_tz_offset

If nonzero, enables *args.tz_offset*.

`args.tz_offset`

A value, in seconds, to be added to the current time in GMT to produce a desired local time. Enabled by *args.specify_tz_offset*. If disabled, the function uses the local time set on the host computer.

`args.fallback_rtp_reinvite`

Specifies whether or not a SIP RTP reINVITE should be transmitted for G.711 fallback mode if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable). Valid values are:

BT_FALLBACK_RTP_REINVITE_DEFAULT

Default to the setting specified in the call control configuration file.

BT_FALLBACK_RTP_REINVITE_DISABLE

Do not transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

BT_FALLBACK_RTP_REINVITE_ENABLE

Transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

When the application sets the value in this field to *BT_FALLBACK_RTP_REINVITE_DEFAULT*, the functionality will default to the setting specified in the call control configuration file (*callctrl.cfg*) by the *g711_fallback_rtp_reinvite* parameter. If the *g711_fallback_rtp_reinvite* parameter isn't specified in the call control configuration file, then the default functionality is *BT_FALLBACK_RTP_REINVITE_DISABLE*.

An application can override the value specified in the call control configuration file by the *g711_fallback_rtp_reinvite* parameter by setting this field to a value of either *BT_FALLBACK_RTP_REINVITE_DISABLE* or *BT_FALLBACK_RTP_REINVITE_ENABLE*.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_DISABLE* will prevent transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_ENABLE* will result in transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable) and the fax transport protocol (*fax_transport_protocol*) parameter specified in the call control configuration file is set to *t38_first*. The SDP settings in the SIP RTP reINVITE will be the same RTP codec settings initially used to establish the call.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol or PSTN line types.

`args.ecm_override`

If set to 0, there is no effect; if > 0, ECM is enabled, overriding the setting of ecm_enable in the user configuration file; if < 0, ECM is disabled, overriding the settings of ecm_enable and t34_enable in the user configuration file. See the ecm_enable parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_restrict_res`

If non-zero, args.restrict_res is active.

`args.restrict_res`

If args.override_restrict_res is non-zero, the specified value overrides the value of restrict_res specified in the user configuration file. See the restrict_res parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_eff_pt_caps`

If non-zero, args.eff_pt_caps is active.

`args.eff_pt_caps`

If args.override_eff_pt_caps is non-zero, the specified value overrides the value of eff_pt_caps specified in the user configuration file. See the eff_pt_caps parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_max_width`

If non-zero, args.max_width is active.

`args.max_width`

If args.override_max_width is non-zero, the specified value overrides the value of max_width specified in the user configuration file. See the max_width parameter in *Volume 6, Appendix A, User-defined Configuration File*.

`args.v34_override`

If set to 0, there is no effect; if > 0, V.34 is enabled, overriding the setting of v34_enable in the user configuration file; if < 0, V.34 is disabled, overriding the settings of v34_enable in the user configuration file. See the v34_enable parameter in *Volume 6, Appendix A, User-defined Configuration File*.

`args.override_line_compression`

If non-zero, args.line_compression is active.

`args.line_compression`

If args.override_line_compression is non-zero, the specified value overrides the value of line_compression specified in the user configuration file. See the line_compression parameter in *Volume 6, Appendix A, User-defined Configuration File*.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

Use this function instead of *BfvFaxBeginSend* or *BfvFaxBeginReceive* when polling is required or permitted. This function handles all polling variations (dial and send, answer and receive, dial and receive, answer and send, dial and send-receive, answer and receive-send).

The channel is initially in send mode after originating (*args.oa_flag* = `BT_ORIGINATE`) a call and initially in receive mode after answering (*args.oa_flag* = `BT_ANSWER`) a call. This initial mode of transfer may change and reverse direction, depending on the *args.s_ips* and *args.r_ips* parameters. When the direction of transfer is reversed, the line state changes to `LINE_STATE_TURNAROUND`. This change in the line state is seen either on return from the first call to

BfvFaxWaitForTraining, before any data is transmitted or received, or on return from *BfvFaxEndOfDocument*, *BfvFaxReceivePages*, or *BfvFaxEndReception*.

The application must keep track of the mode of transfer and, depending on the current direction of transfer, call the appropriate fax send or receive functions.

See the chart under the *BfvFaxPoll* function for information on how to configure *args.s_ips* and *args.r_ips* to set the send/receive mode.

If *args.s_ips* is not `NULL`, the first infopkt must be `INFOPKT_DOCUMENT_PARAMETERS`.

This function sets parameters based on the infopkts appearing at the start of *args.s_ips* that indicate a `beginning-of-page`. Infopkts which indicate a `beginning-of-page` are:

```
INFOPKT_BEGINNING_OF_PAGE
INFOPKT_DOCUMENT_PARAMETERS
INFOPKT_T30_PARAMETERS
INFOPKT_ASCII_PAGE_PARAMETERS
INFOPKT_PAGE_PARAMETERS
INFOPKT_FAX_HDR
INFOPKT_EFF_PAGE_PARAMETERS
```

This function is incorporated into the higher-level function *BfvFaxPoll*.

When calling the *BfvFaxSetLocalId* function, call it before issuing *BfvFaxBegin*.

Before calling this function, the application can use the `LINE_FAX_RES` macro to enable `PAGE_RES` structure allocation.

See Also

BfvFaxBeginTiff, *BfvFaxEndOfDocument*, *BfvFaxEndReception*, *BfvFaxGetRemoteInfo*, *BfvFaxPoll*

Example

```
BTLINE *lp;
struct args_infopkt args_infopkt;
struct args_fax args_fax;

BT_ZERO(args_infopkt);
BT_ZERO(args_fax);
args_infopkt.fname = "report.fax";
args_infopkt.fmode = "r";
args_fax.r_ips = BfvInfopktOpen(&args_infopkt);
args_fax.s_ips = NULL;
args_fax.oa_flag = BT_ANSWER;
/* This performs ordinary Answer and Receive with no
   polling */
BfvFaxBegin(lp, &args_fax);
```

BfvFaxBeginRaw

Purpose

Transmits or receives raw data files.

Syntax

```
void
BfvFaxBeginRaw              (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
int oa_flag;
int can_send;
int can_receive;
int resolution;
int width;
int xmit_mode;
int rcv_mode;
unsigned long eff_page_type;
int specify_tz_offset;
int tz_offset;
unsigned fallback_rtp_reinvite;
int ecm_override;
int override_restrict_res;
int restrict_res;
int override_eff_pt_caps;
int eff_pt_caps;
int override_max_width;
int max_width;
int v34_override;
int override_line_compression;
int line_compression;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.oa_flag

Indicates whether the channel dialed a number or answered a call.

BT_ORIGINATEChannel dialed a number.

BT_ANSWERChannel answered a call.

args.can_send

Indicates whether the channel is enabled to send.

1 Sending enabled.

0 Sending disabled.

args.can_receive

Indicates whether the channel is enabled to receive.

1 Receiving enabled.

0 Receiving disabled.

args.resolution

Resolution of the first page to be transmitted.

See the *BfvFaxBeginSendRaw* function for resolution values.

args.width

Horizontal width of the first page to be transmitted.

See the *BfvFaxBeginSendRaw* function for width values.

args.xmit_mode

Specifies the fax transmit mode. Valid values are:

XMIT_MODE_AUTO 0

Auto transmit mode; standard, default.

XMIT_MODE_MANUAL 1

Manual transmit mode; same as AUTO, but has timing differences when retransmitting FSK signals.

`args.rcv_mode`

Specifies the fax receive mode. Valid values are:

`RCV_MODE_AUTO` 0
Auto receive mode; standard, default.

`RCV_MODE_MANUAL` 1
Manual receive mode; same as `AUTO`, but no initial CED is played and startup delays are reduced.

`args.eff_page_type`

Not supported on TruFax®.

If nonzero, the next transmitted page will be an enhanced fax format page of the specified type. Valid values are:

`BT_EFF_JPEG_ENABLE` 0x1
JPEG. May also logically OR in the following.

`BT_EFF_JPEG_FULLCOLOR` 0x2
Full color.

`BT_EFF_JPEG_DEFAULT_TABLES` 0x4
Default Huffman Tables.

`BT_EFF_JPEG_12BIT` 0x8
12 bits/pel, Otherwise 8.

`BT_EFF_JPEG_NOSUBSAMPLING` 0x10
No subsampling.

`BT_EFF_JPEG_CUSTOM_ILLUMINANT` 0x20
Custom Illuminant.

`BT_EFF_JPEG_CUSTOM_GAMUT` 0x40
Custom Gamut.

`BT_EFF_JBIG` 0x0100
JBIG. May also logically OR in the following.

`BT_EFF_JBIG_L0` 0x0200
L0 Mode.

`args.specify_tz_offset`

If nonzero, enables *args.tz_offset*.

`args.tz_offset`

A value, in seconds, to be added to the current time in GMT to produce a desired local time. Enabled by *args.specify_tz_offset*. If disabled, the function uses the local time set on the host computer.

`args.fallback_rtp_reinvite`

Specifies whether or not a SIP RTP reINVITE should be transmitted for G.711 fallback mode if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable). Valid values are:

BT_FALLBACK_RTP_REINVITE_DEFAULT

Default to the setting specified in the call control configuration file.

BT_FALLBACK_RTP_REINVITE_DISABLE

Do not transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

BT_FALLBACK_RTP_REINVITE_ENABLE

Transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

When the application sets the value in this field to *BT_FALLBACK_RTP_REINVITE_DEFAULT*, the functionality will default to the setting specified in the call control configuration file (*callctrl.cfg*) by the *g711_fallback_rtp_reinvite* parameter. If the *g711_fallback_rtp_reinvite* parameter isn't specified in the call control configuration file, then the default functionality is *BT_FALLBACK_RTP_REINVITE_DISABLE*.

An application can override the value specified in the call control configuration file by the *g711_fallback_rtp_reinvite* parameter by setting this field to a value of either *BT_FALLBACK_RTP_REINVITE_DISABLE* or *BT_FALLBACK_RTP_REINVITE_ENABLE*.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_DISABLE* will prevent transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_ENABLE* will result in transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable) and the fax transport protocol (*fax_transport_protocol*) parameter specified in the call control configuration file is set to *t38_first*. The SDP settings in the SIP RTP reINVITE will be the same RTP codec settings initially used to establish the call.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol or PSTN line types.

`args.ecm_override`

If set to 0, there is no effect; if > 0, ECM is enabled, overriding the setting of ecm_enable in the user configuration file; if < 0, ECM is disabled, overriding the settings of ecm_enable and t34_enable in the user configuration file. See the ecm_enable parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_restrict_res`

If non-zero, args.restrict_res is active.

`args.restrict_res`

If args.override_restrict_res is non-zero, the specified value overrides the value of restrict_res specified in the user configuration file. See the restrict_res parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_eff_pt_caps`

If non-zero, args.eff_pt_caps is active.

`args.eff_pt_caps`

If args.override_eff_pt_caps is non-zero, the specified value overrides the value of eff_pt_caps specified in the user configuration file. See the eff_pt_caps parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_max_width`

If non-zero, args.max_width is active.

`args.max_width`

If args.override_max_width is non-zero, the specified value overrides the value of max_width specified in the user configuration file. See the max_width parameter in *Volume 6, Appendix A, User-defined Configuration File*.

`args.v34_override`

If set to 0, there is no effect; if > 0, V.34 is enabled, overriding the setting of v34_enable in the user configuration file; if < 0, V.34 is disabled, overriding the settings of v34_enable in the user configuration file. See the v34_enable parameter in *Volume 6, Appendix A, User-defined Configuration File*.

`args.override_line_compression`

If non-zero, args.line_compression is active.

`args.line_compression`

If args.override_line_compression is non-zero, the specified value overrides the value of line_compression specified in the user configuration file. See the line_compression parameter in *Volume 6, Appendix A, User-defined Configuration File*.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

Initiates the fax process in originate or answer mode depending on the contents of the *args.oa_flag*.

Use this function instead of *BfvFaxBeginSendRaw* or *BfvFaxBeginReceive* when polling is required or permitted. See the *BfvFaxBegin* and *BfvFaxPoll* functions for more information on polling.

This function handles all polling variations (dial and send, answer and receive, dial and receive, answer and send, dial and send-receive, answer and receive-send).

The *BfvFaxSetLocalId*, *BfvFaxPageParams* or *BfvFaxT30Params* functions can be called before this function.

Before calling this function, the application can use the LINE_FAX_RES macro to enable PAGE_RES structure allocation.

Many fax devices do not support the reception of enhanced fax format pages. To ensure that the receiver can support the format you wish to send, you should use the T.30 holdup feature and examining the DIS (see *BfvFaxT30Holdup* and *LINE_DIS_DTC (lp)*).

See Also

BfvFaxBegin, *BfvFaxBeginTiff*

Example

struct args_fax args;

```
/* performs ordinary Answer and Receive with no polling */
BT_ZERO(args);
args.oa_flag = BT_ANSWER;
args.can_send = 0;
args.can_receive = 1;
args.resolution = RES_200H_100V;
args.width = WIDTH_A4;
args.rcv_mode = RCV_MODE_AUTO;
BfvFaxBeginRaw (lp, &args);
```

BfvFaxBeginReceive

Purpose

Instructs the channel to begin fax reception.

Syntax

```
int
BfvFaxBeginReceive          (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
int rcv_mode;
unsigned fallback_rtp_reinvite;
int ecm_override;
int override_restrict_res;
int restrict_res;
int override_eff_pt_caps;
int eff_pt_caps;
int override_max_width;
int max_width;
int v34_override;
int override_line_compression;
int line_compression;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

`args.rcv_mode`

Specifies the fax receive mode. Valid values are:

`RCV_MODE_AUTO` 0

Auto receive mode; standard, default.

`RCV_MODE_MANUAL` 1

Manual receive mode; same as `AUTO`, but no initial CED is played and startup delays are reduced.

`args.fallback_rtp_reinvite`

Specifies whether or not a SIP RTP reINVITE should be transmitted for G.711 fallback mode if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable). Valid values are:

BT_FALLBACK_RTP_REINVITE_DEFAULT

Default to the setting specified in the call control configuration file.

BT_FALLBACK_RTP_REINVITE_DISABLE

Do not transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

BT_FALLBACK_RTP_REINVITE_ENABLE

Transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

When the application sets the value in this field to *BT_FALLBACK_RTP_REINVITE_DEFAULT*, the functionality will default to the setting specified in the call control configuration file (*callctrl.cfg*) by the *g711_fallback_rtp_reinvite* parameter. If the *g711_fallback_rtp_reinvite* parameter isn't specified in the call control configuration file, then the default functionality is *BT_FALLBACK_RTP_REINVITE_DISABLE*.

An application can override the value specified in the call control configuration file by the *g711_fallback_rtp_reinvite* parameter by setting this field to a value of either *BT_FALLBACK_RTP_REINVITE_DISABLE* or *BT_FALLBACK_RTP_REINVITE_ENABLE*.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_DISABLE* will prevent transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_ENABLE* will result in transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable) and the fax transport protocol (*fax_transport_protocol*) parameter specified in the call control configuration file is set to *t38_first*. The SDP settings in the SIP RTP reINVITE will be the same RTP codec settings initially used to establish the call.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol or PSTN line types.

`args.ecm_override`

If set to 0, there is no effect; if > 0, ECM is enabled, overriding the setting of ecm_enable in the user configuration file; if < 0, ECM is disabled, overriding the settings of ecm_enable and t34_enable in the user configuration file. See the ecm_enable parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_restrict_res`

If non-zero, args.restrict_res is active.

`args.restrict_res`

If args.override_restrict_res is non-zero, the specified value overrides the value of restrict_res specified in the user configuration file. See the restrict_res parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_eff_pt_caps`

If non-zero, args.eff_pt_caps is active.

`args.eff_pt_caps`

If args.override_eff_pt_caps is non-zero, the specified value overrides the value of eff_pt_caps specified in the user configuration file. See the eff_pt_caps parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_max_width`

If non-zero, args.max_width is active.

`args.max_width`

If args.override_max_width is non-zero, the specified value overrides the value of max_width specified in the user configuration file. See the max_width parameter in *Volume 6, Appendix A, User-defined Configuration File*.

`args.v34_override`

If set to 0, there is no effect; if > 0, V.34 is enabled, overriding the setting of v34_enable in the user configuration file; if < 0, V.34 is disabled, overriding the settings of v34_enable in the user configuration file. See the v34_enable parameter in *Volume 6, Appendix A, User-defined Configuration File*.

`args.override_line_compression`

If non-zero, args.line_compression is active.

`args.line_compression`

If args.override_line_compression is non-zero, the specified value overrides the value of line_compression specified in the user configuration file. See the line_compression parameter in *Volume 6, Appendix A, User-defined Configuration File*.

Output

Return value:

0	Channel was successfully set at the beginning of the T.30 sequence.
<0	Failure to set channel at the beginning of the T.30 sequence.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

Use this function to initiate reception of any kind of fax data (infopacket, raw, TIFF, etc.). The function places the channel at the beginning of Phase B of the T.30 sequence to initiate transmission of the `DIS` to the calling machine.

If the application intends to call the *BfvFaxSetLocalId* function, it should do so before calling this function.

Call *BfvFaxGetRemoteInfo* next to set the channel to wait for information from the remote machine.

Checks to ensure that the line state is set to CONNECTED.

Polling cannot occur when the application uses this function. When the application uses low-level infopkt functions, polling can occur only with the *BfvFaxBegin* function. When the application uses noninfopkt, raw fax data functions, polling can occur with the *BfvFaxBeginRaw* function. When the application uses TIFF-F functions, polling can occur with the *BfvFaxBeginTiff* function.

This function is incorporated into the higher-level function *BfvFaxReceive*.

The application can call the *BfvFaxSetReceiveFmt* function before calling this function.

Before calling this function, the application can use the LINE_FAX_RES macro to enable PAGE_RES structure allocation.

See Also

BfvFaxGetRemoteInfo, *BfvFaxReceive*

Example

```
BTLINE *lp;
struct args_fax args;

BT_ZERO(args);
if (BfvFaxBeginReceive(&args) < 0)
{
    printf("Error in BfvFaxBeginReceive\n");
}
```

BfvFaxBeginSend

Purpose

Prepares the line data structure for fax transmission.

Syntax

```
void
BfvFaxBeginSend              (lp, args)
   BTLINE                    *lp;
   struct args_fax           *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *s_ips;
int xmit_mode;
int specify_tz_offset;
int tz_offset;
int ecm_override;
```

Output Fields

```
RES res;
```

Modified Fields

resolution, width, eff_page_type

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.s_ips

Pointer to the infopkt stream to transmit.

`args.xmit_mode`

Specifies the fax transmit mode. Valid values are:

`XMIT_MODE_AUTO` 0
: Auto transmit mode; standard, default.

`XMIT_MODE_MANUAL` 1
: Manual transmit mode; same as `AUTO`, but has timing differences when retransmitting FSK signals.

`args.specify_tz_offset`

If nonzero, enables *args.tz_offset*.

`args.tz_offset`

A value, in seconds, to be added to the current time in GMT to produce a desired local time. Enabled by *args.specify_tz_offset*. If disabled, the function uses the local time set on the host computer.

`args.ecm_override`

If set to 0, there is no effect; if > 0, ECM is enabled, overriding the setting of ecm_enable in the user configuration file; if < 0, ECM is disabled, overriding the settings of ecm_enable and t34_enable in the user configuration file. See the ecm_enable parameter in *Volume 6, Appendix A, User-defined Configuration File.*

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function begins fax transmission using an infopkt stream (places the channel at the top of Phase B on the transmit side to look for a `DIS` signal from the answering fax machine).

If you call the *BfvFaxSetLocalId* function, call it before calling this function.

The first infopkt in *ips* must be:

`INFOPKT_DOCUMENT_PARAMETERS`

This function sets parameters based on the infopkts appearing at the start of *args.s_ips* that indicate a `beginning-of-page`. Infopkts that indicate a `beginning-of-page` are:

```
INFOPKT_BEGINNING_OF_PAGE
INFOPKT_DOCUMENT_PARAMETERS
INFOPKT_T30_PARAMETERS
INFOPKT_ASCII_PAGE_PARAMETERS
INFOPKT_PAGE_PARAMETERS
INFOPKT_FAX_HDR
INFOPKT_EFF_PAGE_PARAMETERS
```

This function is incorporated into the higher-level function *BfvFaxSend*.

Polling cannot occur when the application uses this function. When the application uses low-level functions, polling can only occur with the *BfvFaxBegin* function.

Before calling this function, the application can use the `LINE_FAX_RES` macro to enable `PAGE_RES` structure allocation.

See Also

BfvFaxBeginSendRaw, *BfvFaxBeginSendTiff*, *BfvFaxGetRemoteInfo*, *BfvFaxSend*, *BfvFaxSendPage*, `LINE_FAX_RES`

Example

```
BTLINE *lp;
struct infopkt_stream *ips;
struct args_fax args;

BT_ZERO(args);
args.s_ips = ips;
BfvFaxBeginSend(lp,&args);
if (args.res.status != BT_STATUS_OK)
{
    printf ("Error in BfvFaxBeginSend\n");
}
```

BfvFaxBeginSendRaw

Purpose

Prepares the line data structure to transmit raw data files.

Syntax

```
void
BfvFaxBeginSendRaw          (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
int resolution;
int width;
int xmit_mode;
unsigned long eff_page_type;
int specify_tz_offset;
int tz_offset;
int ecm_override;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.resolution

`RES_200H_400V`, `RES_300H_300V`, `RES_400H_400V`, `RES_600H_600V`, `RES_1200H_1200V`, `RES_300H_600V`, `RES_400H_800V`, `RES_600H_1200V`, `RES_100H_100V`, are not supported on TruFax®.

Resolution of the first page of the transmitting document. Values are:

`RES_200H_100V` 0
: 204 dpi horizontal, 98 dpi vertical (normal resolution, also `RES_NORMAL`)

`RES_200H_200V` 1
: 204 dpi horizontal, 196 dpi vertical (fine resolution, also `RES_FINE`)

`RES_200H_400V` 2
: 200 dpi horizontal, 400 dpi vertical

`RES_300H_300V` 3
: 300 dpi horizontal, 300 dpi vertical

`RES_400H_400V` 4
: 400 dpi horizontal, 400 dpi vertical

`RES_600H_600V` 5
: 600 dpi horizontal, 600 dpi vertical

`RES_1200H_1200V` 6
: 1200 dpi horizontal, 1200 dpi vertical

`RES_300H_600V` 7
: 300 dpi horizontal, 600 dpi vertical

`RES_400H_800V` 8
: 400 dpi horizontal, 800 dpi vertical

`RES_600H_1200V` 9
: 600 dpi horizontal, 1200 dpi vertical

`RES_100H_100V` 10
: 100 dpi horizontal, 100 dpi vertical. For use with JPEG only.

Note: Non-square pixels cannot be used for JPEG.

`args.width`

Horizontal width of the first page to be transmitted. Valid values are:

`WIDTH_A4`0: 215 mm, 1728 normal resolution pixels.

`WIDTH_B4`1: 255 mm, 2048 normal resolution pixels.

`WIDTH_A3`2: 303 mm, 2432 normal resolution pixels.

`args.xmit_mode`

Specifies the fax transmit mode. Valid values are:

`XMIT_MODE_AUTO` 0
: Auto transmit mode; standard, default.

`XMIT_MODE_MANUAL` 1
: Manual transmit mode; same as `AUTO`, but has timing differences when retransmitting FSK signals.

`args.eff_page_type`

Not supported on TruFax®.

If nonzero, the next transmitted page will be an enhanced fax format page of the specified type. Valid values are:

`BT_EFF_JPEG_ENABLE` 0x1
: JPEG. May also logically OR in the following.

`BT_EFF_JPEG_FULLCOLOR` 0x2
: Full color.

`BT_EFF_JPEG_DEFAULT_TABLES` 0x4
: Default Huffman Tables.

`BT_EFF_JPEG_12BIT` 0x8
: 12 bits/pel, Otherwise 8.

`BT_EFF_JPEG_NOSUBSAMPLING` 0x10
: No subsampling.

`BT_EFF_JPEG_CUSTOM_ILLUMINANT` 0x20
: Custom Illuminant.

`BT_EFF_JPEG_CUSTOM_GAMUT` 0x40
: Custom Gamut.

`BT_EFF_JBIG` 0x0100
: JBIG. May also logically OR in the following.

`BT_EFF_JBIG_L0` 0x0200
: L0 Mode.

`args.specify_tz_offset`

If nonzero, enables *args.tz_offset*.

`args.tz_offset`

A value, in seconds, to be added to the current time in GMT to produce a desired local time. Enabled by *args.specify_tz_offset*. If disabled, the function uses the local time set on the host computer.

`args.ecm_override`

If set to 0, there is no effect; if > 0, ECM is enabled, overriding the setting of ecm_enable in the user configuration file; if < 0, ECM is disabled, overriding the settings of ecm_enable and t34_enable in the user configuration file. See the ecm_enable parameter in *Volume 6, Appendix A, User-defined Configuration File.*

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function begins fax transmission of raw data files (places the channel at the top of Phase B on the transmit side to look for a `DIS` signal from the answering fax machine).

The *BfvFaxSetLocalId*, *BfvFaxPageParams* or *BfvFaxT30Params* functions can be called before this function.

Polling cannot occur when the application uses this function. When the application uses noninfopkt raw data functions, polling can occur only with the *BfvFaxBeginRaw* function.

Before calling this function, the application can use the `LINE_FAX_RES` macro to enable `PAGE_RES` structure allocation.

Many fax devices do not support the reception of enhanced fax format pages. To ensure that the receiver can support the format you wish to send, you should use the T.30 holdup feature and examining the DIS (see *BfvFaxT30Holdup* and *LINE_DIS_DTC (lp)*).

See Also

BfvFaxBeginSend, *BfvFaxBeginSendTiff*, *BfvFaxGetRemoteInfo*, `LINE_FAX_RES`

Example

```
BTLINE *lp;
struct args_fax args;

BT_ZERO(args);
args.resolution = RES_200H_200V;/* fine resolution first
                                   page */
args.width = WIDTH_A4;          /* A4 width 1st page */
BfvFaxBeginSendRaw(lp, &args);
```

BfvFaxBeginSendTiff

Purpose

Prepares the line data structure to transmit TIFF-F files.

Syntax

```
void
BfvFaxBeginSendTiff          (lp, args)
   BTLINE                    *lp;
   struct args_fax           *args;
```

The structure contains the following fields.

Input Fields

```
TFILE *s_tp;
int xmit_mode;
int specify_tz_offset;
int tz_offset;
int ecm_override;
```

Output Fields

```
RES res;
```

Modified Fields

resolution, width

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.s_tp

*TFILE** pointer to a TIFF file ready for transmission.

args.xmit_mode

Specifies the fax transmit mode. Valid values are:

XMIT_MODE_AUTO Auto transmit mode; standard, default.	0
XMIT_MODE_MANUAL Manual transmit mode; same as AUTO, but has timing differences when retransmitting FSK signals.	1

args.specify_tz_offset

If nonzero, enables *args.tz_offset*.

args.tz_offset

A value, in seconds, to be added to the current time in GMT to produce a desired local time. Enabled by *args.specify_tz_offset*. If disabled, the function uses the local time set on the host computer.

args.ecm_override

If set to 0, there is no effect; if > 0, ECM is enabled, overriding the setting of ecm_enable in the user configuration file; if < 0, ECM is disabled, overriding the settings of ecm_enable and t34_enable in the user configuration file. See the ecm_enable parameter in *Volume 6, Appendix A, User-defined Configuration File.*

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function begins fax transmission of TIFF-F files (places the channel at the top of Phase B on the transmit side to look for a DIS signal from the answering fax machine).

The *BfvFaxSetLocalId*, *BfvFaxPageParams* or *BfvFaxT30Params* functions can be called before this function.

Before calling this function, the application can use the LINE_FAX_RES macro to enable PAGE_RES structure allocation.

If sending is enabled using *args.s_tp*, the application can send other non-TIFF G3 or ASCII data for the channel to combine on the same page after calling this function. The application will then call *BfvFaxSendPageTiff* at the appropriate time. This sequencing permits the application to use the width and resolution from the TIFF file for the page and for the individual data strip.

TIFF-F, as supported by Dialogic® Brooktrout® boards, only supports MH, MR, and MMR. It does not support JPEG, JBIG, or any other enhanced fax formats.

See Also

BfvFaxBegin, LINE_FAX_RES

Example

```
BTLINE *lp;
TFILE *tp;
struct args_fax args;

BT_ZERO(args);
args.s_tp = tp;
BfvFaxBeginSendTiff(lp, &args);
```

BfvFaxBeginTiff

Purpose

Initiates the fax transmission process of TIFF-F files in originate or answer mode depending on whether the channel dialed a number or answered a call.

Syntax

```
void
BfvFaxBeginTiff              (lp, args)
   BTLINE                    *lp;
   struct args_fax           *args;
```

The structure contains the following fields.

Input Fields

```
int oa_flag;
TFILE *s_tp;
TFILE *r_tp;
int xmit_mode;
int rcv_mode;
int specify_tz_offset;
int tz_offset;
unsigned fallback_rtp_reinvite;
int ecm_override;
```

Output Fields

```
RES res;
```

Modified Fields

can_send, can_receive, resolution, width

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.oa_flag

Indicates whether the channel dialed a number or answered a call.

`BT_ORIGINATE` Channel dialed a number.

`BT_ANSWER` Channel answered a call.

args.s_tp

*TFILE** pointer to a TIFF file ready for transmission.

`NULL` = Sending is disabled.

args.r_tp

*TFILE** pointer to a TIFF file ready for reception.

`NULL` = Receiving is disabled.

args.xmit_mode

Specifies the fax transmit mode. Valid values are:

`XMIT_MODE_AUTO` 0

Auto transmit mode; standard, default.

`XMIT_MODE_MANUAL` 1

Manual transmit mode; same as `AUTO`, but has timing differences when retransmitting FSK signals.

args.rcv_mode

Specifies the fax receive mode. Valid values are:

`RCV_MODE_AUTO` 0

Auto receive mode; standard, default.

`RCV_MODE_MANUAL` 1

Manual receive mode; same as `AUTO`, but no initial CED is played and startup delays are reduced.

args.specify_tz_offset

If nonzero, enables *args.tz_offset*.

`args.tz_offset`

A value, in seconds, to be added to the current time in GMT to produce a desired local time. Enabled by *args.specify_tz_offset*. If disabled, the function uses the local time set on the host computer.

`args.fallback_rtp_reinvite`

Specifies whether or not a SIP RTP reINVITE should be transmitted for G.711 fallback mode if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable). Valid values are:

BT_FALLBACK_RTP_REINVITE_DEFAULT

Default to the setting specified in the call control configuration file.

BT_FALLBACK_RTP_REINVITE_DISABLE

Do not transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

BT_FALLBACK_RTP_REINVITE_ENABLE

Transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

When the application sets the value in this field to *BT_FALLBACK_RTP_REINVITE_DEFAULT*, the functionality will default to the setting specified in the call control configuration file (*callctrl.cfg*) by the *g711_fallback_rtp_reinvite* parameter. If the *g711_fallback_rtp_reinvite* parameter isn't specified in the call control configuration file, then the default functionality is *BT_FALLBACK_RTP_REINVITE_DISABLE*.

An application can override the value specified in the call control configuration file by the *g711_fallback_rtp_reinvite* parameter by setting this field to a value of either *BT_FALLBACK_RTP_REINVITE_DISABLE* or *BT_FALLBACK_RTP_REINVITE_ENABLE*.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_DISABLE* will prevent transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_ENABLE* will result in transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable) and the fax transport protocol (*fax_transport_protocol*) parameter specified in the call control configuration file is set to *t38_first*. The SDP settings in the SIP RTP reINVITE will be the same RTP codec settings initially used to establish the call.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol or PSTN line types.

`args.ecm_override`

If set to 0, there is no effect; if > 0, ECM is enabled, overriding the setting of ecm_enable in the user configuration file; if < 0, ECM is disabled, overriding the settings of ecm_enable and t34_enable in the user configuration file. See the ecm_enable parameter in *Volume 6, Appendix A, User-defined Configuration File.*

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function transmits TIFF-F files in originate or answer mode depending on the contents of the *args.oa_flag*. The function handles all polling variations (dial and send, answer and receive, dial and receive, answer and send, dial and send and receive, answer and receive and send).

Use this function instead of *BfvFaxBeginSendTiff* or *BfvFaxBeginReceive* when polling is required or permitted. See the *BfvFaxBegin* and *BfvFaxPoll* functions for more information on polling.

The *BfvFaxSetLocalId* or *BfvFaxPageParams* functions can be called before this function, and the *BfvFaxT30Params* function can be called immediately after this function.

Before calling this function, the application can use the `LINE_FAX_RES` macro to enable `PAGE_RES` structure allocation.

If sending is enabled using *args.s_tp*, the application can send other non-TIFF G3 or ASCII data for the channel to combine on the same page after calling this function. The application will then call *BfvFaxSendPageTiff* at the appropriate time. This sequencing permits the application to use the width and resolution from the TIFF file for the page and for the individual data strip.

TIFF-F, as supported by Dialogic® Brooktrout® boards, only supports MH, MR, and MMR. It does not support JPEG, JBIG, or any other enhanced fax formats.

See Also

BfvFaxBegin, *BfvFaxBeginRaw*, LINE_FAX_RES

Example

```
/* Performs ordinary Answer and Receive with */
/* no polling */
BTLINE *lp;
TFILE *tp = BfvTiffOpen("fax.tiff", "w");
struct args_fax args;

BT_ZERO(args);
args.oa_flag = BT_ANSWER;
args.s_tp = NULL;
args.r_tp = tp;
BfvFaxBeginTiff(lp, &args);
```

BfvFaxDownloadFont

Purpose

Downloads the indicated font file to the channel for use as the specified numbered font.

Syntax

```
int
BfvFaxDownloadFont          (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
int font_no;
```

Output Fields

```
RES res;
```

Modified Fields

buf, size.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

`args.fname`

The name of the font file to download to the channel.

The following font files are currently available in the fonts directory:

Normal Resolution	Fine Resolution
epsones.fz8	*courb8.fz8*
epsonps.fz8	*courb10.fz8*
epsonpc.fz8	*courb18.fz8*
epsonec.fz8	*wcrb8.fz8*
ibmpcs.fz8	

`args.font_no`

The font number to download. For more information see the *font_file* parameter in *Volume 6, Appendix A*.

The range is 0-6 or 255.

Output

Return value:

0 Font file successfully downloaded.

<0 Failure to download the font file.

Failure to download a font file could occur due to one of the following reasons: failure to find the file, failure to open the file, or failure to allocate memory.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The fonts specified by the *font_file* parameters in the user-defined configuration file are automatically downloaded to the module when the function is called. Use the *LINE_FONT_DOWNLOADED (lp, font_no)* macro to determine if a font was successfully downloaded.

The number of fonts that a module can support is hardware and firmware dependent. No more than seven fonts (0-6) per module can be downloaded. These fonts will be shared across all channels on the module. Total memory for font storage is fixed; see the Firmware Release Notes for the amount of memory.

Only line pointers attached to logical channel 1 using *BfvSessionAttach* can download fonts. The fonts apply to all channels on a module. Once a font with a given number is downloaded, it cannot be replaced without re-initializing the module. The font number 255 serves as a default font, which is used if a font number referenced for ASCII conversion has not been loaded.

If downloading a font, the application must do it before initiating a call or waiting for ring. Some font numbers may be reserved for preloaded fonts.

The normal resolution font sets currently available from Dialogic emulate either the Epson LX or the IBM extended character set and are set in pica (10 characters/inch), elite (12 characters/inch), or compressed (132 characters/line) format. The normal resolution font sets are:

Epson LX type:	*epsonps.fz8*	(Pica standard)
	epsonpc.fz8	(Pica compressed)
	epsones.fz8	(Elite standard)
	epsonec.fz8	(Elite compressed)
IBM PC type:	*ibmpcps.fz8*	(Pica standard)

The fine resolution font sets currently available from Dialogic are set in courier bold in 8, 10, and 18 points. The fine resolution font sets are:

IBM PC type:	*courb8.fz8*	(8 points; 256 chars., 0-255)
	courb10.fz8	(10 points; 137 chars., 32-168)
	courb18.fz8	(18 points; 64 chars., 32-95)
Windows type:	*wcrb8.fz8*	(8 points; 256 chars., 0-255)

Note: If the firmware encounters an ASCII character that is not a member of the current font, the firmware replaces the character with a "space" (ASCII 32) character.

See Also

BfvFaxDownloadFontData, *BfvLineReset*, LINE_FONT_DOWNLOADED

Example

```
BTLINE *lp;
struct args_fax args;

BT_ZERO(args);
args.fname = "ibmpcps.fz8";
args.font_no = 0;
BfvFaxDownloadFont(lp, &args)
```

BfvFaxDownloadFontData

Purpose

Downloads the font supplied in a user-allocated data buffer to the module for use as the specified numbered font.

Syntax

```
int
BfvFaxDownloadFontData      (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
unsigned char *buf;
unsigned size;
int font_no;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.buf

Pointer to a user-allocated data buffer that contains the font data to download.

args.size

Size, in bytes, of the user-allocated data buffer.

args.font_no

The number of the font to download. The range is 0-6 or 255.

Output

Return value:

0 Font file successfully downloaded.

<0 Font file failed to download.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function must be called repeatedly with buffers of data until the entire font is downloaded. After all data are downloaded, *BfvFaxDownloadFontData* must be called once with an *args.buf* value of `NULL` and an *args.size* value of 0.

The fonts specified by the *font_file* parameters in the user-defined configuration file are automatically downloaded to the module when the *BfvLineReset* function is called.

See the *LINE_FONT_DOWNLOADED (lp, font_no)* macro to determine if a font was successfully downloaded.

The number of fonts that can be supported by a module is hardware and firmware dependent. No more than seven fonts (0-6) per module can be downloaded. These fonts will be shared across all channels on the module. Total memory for font storage is fixed; see the Release Notes for the amount of memory.

Only line pointers attached to logical channel 1 using *BfvSessionAttach* can download fonts. The fonts apply to all channels on a module. Once a font with a given number is downloaded, it cannot be replaced without re-initializing the module. The font number 255 serves as a default font, which is used if a font number referenced for ASCII conversion has not been loaded.

If downloading a font, the application must do it before initiating a call or waiting for ring. Some font numbers may be reserved for preloaded fonts.

See the *BfvFaxDownloadFont* function for detailed information about the fonts that are supported by Dialogic.

See Also

BfvFaxDownloadFont, *BfvLineReset*, `LINE_FONT_DOWNLOADED`

Example

```
BTLINE *lp;
FILE *fp = fopen("ibmpcps.fz8","r");
unsigned char buf[1024];
unsigned n;
struct args_fax args;

BT_ZERO(args);
while ((n=fread(buf, 1, sizeof(buf), fp)) > 0)
{
    args.buf = buf;
    args.size = n;
    args.font_no = 0;
    BfvFaxDownloadFontData(lp, &args);
}
args.buf = NULL;
args.size = 0;
args.font_no = 0;
BfvFaxDownloadFontData(lp, &args);
```

BfvFaxEndOfDocument

Purpose

Sends the channel an `end-of-page`, accompanied by a flag that indicates no more pages are to follow.

Syntax

```
int
BfvEndOfDocument            (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Output Fields

```
RES res;
```

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

Output

ReturnValue:

<0 Error occurred, transmission not completed successfully.

0 Last page transmitted successfully.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

On return from this function, the last page was sent to the remote fax machine and:

- Either the channel went on-hook, and the line state was changed to `IDLE`

or

- The channel remained off-hook and the line state was changed to `TURNAROUND` if polling was enabled using a function such as *BfvFaxBegin* or *BfvFaxBeginRaw*. The transmitter then becomes a receiver, and *BfvFaxGetRemoteInfo* and *BfvFaxWaitForTraining* must be called again.

This function is incorporated into the higher-level functions *BfvFaxSend* and *BfvFaxPoll*.

Example

```
BTLINE *lp;
struct infopkt_stream *ips;
struct args_fax args;

for (;;)
{
    BT_ZERO(args);
    args.s_ips = ips;
    if (BfvFaxNextPage(lp, &args) <= 0)
        break;
    BT_ZERO(args);
    args.s_ips = ips;
    BfvFaxSendPage(lp, &args);
}
BfvFaxEndOfDocument(lp, &args);
```

BfvFaxEndReception

Purpose

Waits after receiving the last page of a fax transmission for the completion of the T.30 handshaking confirmation sequence.

Syntax

```
int
BfvFaxEndReception          (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return:

<0 Error occurred, reception not completed successfully.

0 Last page received successfully.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this function after receiving the last page of a fax transmission. On the function's return, the last page was received from the remote fax machine and:

- Either the channel went on-hook, and the line state was changed to `IDLE`

or

- The channel remained off-hook and the line state was changed to `TURNAROUND` if polling was enabled using a function such as *BfvFaxBegin* or *BfvFaxBeginRaw*. The receiver then becomes a transmitter, and *BfvFaxGetRemoteInfo* and *BfvFaxWaitForTraining* must be called again.

This function is incorporated into the higher-level function *BfvFaxReceivePages*.

Example

```
BTLINE *lp;
struct infopkt_stream *ips;
struct args_fax args;

do
{
    BT_ZERO(args);
    args.r_ips = ips;
}
while (BfvFaxReceivePage(lp, &args) > 0);
BfvFaxEndReception(lp, &args);
```

BfvFaxGetRemoteInfo

Purpose

Waits until the remote machine sends DIS, DTC, or DCS, and reports the remote ID, NSF/NSS/NSC information, and SUB/PWD/SEP information.

Syntax

```
void
BfvFaxGetRemoteInfo          (lp, args)
   BTLINE                    *lp;
   struct args_fax           *args;
```

The structure contains the following fields.

Output Fields

```
INFO_RES remote_info;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.remote_info

INFO_RES type structure reporting the remote ID and any NSF/NSS/NSC or SUB/PWD/SEP information from the remote fax device. The INFO_RES structure is documented in *Volume 6, Appendix B*.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Before calling this function, the application must call one of the following functions:

BfvFaxBegin
BfvFaxBeginRaw
BfvFaxBeginReceive
BfvFaxBeginSend
BfvFaxBeginSendRaw
BfvFaxBeginSendTiff
BfvFaxBeginTiff

Processes interrupts in a loop until DIS, DTC, or DCS (FSK commands) are received and sets the current line state to `AWAIT_TRAINING`. At each pass through the loop, the line state is tested to ensure that it is still `CONNECTED`.

This function is incorporated into the higher-level functions *BfvFaxSend*, *BfvFaxReceive*, and *BfvFaxPoll*.

Normally *BfvFaxWaitForTraining* is called after this function returns.

When this function returns without error, the line state changes to `AWAIT_TRAINING`.

The format of the `nsf_nss_frame` field in the `INFO_RES` structure is:

Length, data, length, data, ..., etc.

where length (one byte) is the length of the next piece of NSF, NSS, or NSC information, not including the length byte. When length is zero, no more NSF, NSS, or NSC data is available.

See Also

BfvFaxWaitForTraining, `LINE_DCS`, `LINE_DIS_DTC`

Example

```
BTLINE *lp;
struct args_fax args;

BT_ZERO(args);
BfvFaxGetRemoteInfo(lp, &args);
if (args.res.status != BT_STATUS_OK)
    printf("Failed to Get Remote info\n");
```

BfvFaxHeader

Purpose

Sets up a header or footer to appear on all subsequent pages of a fax transmission, using the specified text format and insertion mode.

Syntax

```
void
BfvFaxHeader                (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
int placement;
int insert_mode;
char *label;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.placement

HDR_HEADER
: Specifies a header (top of page).

HDR_FOOTER
: Specifies a footer (bottom of page).

args.insert_mode

HDR_MODE_DISABLE
: Disables the current header.

HDR_MODE_INSERT
: Inserts a header before or after an image.

HDR_MODE_REPLACE
: Replaces an image with a header.

HDR_MODE_OVERLAY
: Forms a bit-wise logical OR of the image and the header.

args.label

A text string that specifies the format of the header or footer using format characters described under *Details*.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

Using the specified text format (*args.label*) and insertion mode (*args.insert_mode*), sets up a header or footer to appear on all subsequent pages of a fax transmission.

Only disable and insert are valid for footers.

When transmitting infopkt files, you can insert the INFOPKT_FAX_HDR infopkt type in the input stream instead of calling this function.

Applications usually call this function once before calling *BfvFaxSend* or *BfvFaxBeginSendRaw*. But, to reset page numbers for example, applications can call it between fax pages during transmission.

The label format uses date/printf style specifications to place items of information into header or footer text lines.

All characters, except ‘%’, represent themselves on output. When the function encounters a ‘%’ character, it bases its output format on the format character immediately following the ‘%’ character.

Note: Some roll paper fax machines may have a gap of about 1/8 inch when printing a fax. This means that the fax machine may not display the first character of the header/footer because the header/footer is positioned at address 0. To avoid this condition, add leading spaces to the fax header/footer.

The format characters include:

a	Day of the week, 3-character abbreviation
A	Day of the week, full-length
b	Month, 3 character abbreviation.
m	Month, 01-12
g	Month, _1-12, a two-character field where a leading zero is replaced by a space. (The underscore ("_") character here is used to represent a space character.)
d	Day of the month, 01-31
e	Day of the month, _1-31, a two-character field where a leading zero is replaced by a space. (The underscore ("_") character here is used to represent a space character.)
H	Hour, 00-23
h	Hour, __-23, a two-character field where leading zeroes are replaced by spaces. (The underscore ("_") character here is used to represent a space character.)
I	Hour, 01-12
M	Minute, 00-59
k	Minute, __-59, a two-character field where leading zeroes are replaced by spaces. (The underscore ("_") character here is used to represent a space character.)
p	Indicates the time of day (AM/PM). If you use this format character, you must also use the “I” hour character.
nP	Page number, 01-99 (leading 0 is rendered as a space), optional ‘n’ resets page numbering as specified, and when sent together, the ‘n’ in the header overrides the ‘n’ in the footer.
nQ	Same as nP, but uses 2-digit page numbers 01-99 with leading zeros.
nU	Same as nP, but uses 3-digit page numbers 001-999 with leading zeros.

nR	Same as nP, but uses 3-digit page numbers 001-999
Nf	Switch to font #N. The range of N is 0-6. The change takes effect from this point in the header/footer. Applications can only mix fonts of the same height. Text preceding the first occurrence of or in absence of this font specification uses a default font. If the specified font has not been downloaded, a default font is used (see *BfvFaxDownloadFont* or the *font_file* keyword in *Volume 6, Appendix A*).
N^	Insert N blank G3 lines before header/footer. This directive can only appear at the beginning of the format string.
N_	Insert N blank G3 lines after header/footer. This directive can only appear at the beginning of the format string.
S	Seconds, 00-59
y	Year, 00-99
Y	4-digit year
%	Percent (%) character

Example

```
struct args_fax args;

BT_ZERO(args);
args.placement = HDR_HEADER;
args.insert_mode = HDR_MODE_INSERT;
args.label = "Time is %H:%M, page %P";
BfvFaxHeader(lp &args);
```

BfvFaxNextPage

Purpose

Sends an `end-of-page` to the channel and processes parameters for the next page of data to be transmitted.

Syntax

```
int
BfvFaxNextPage              (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *s_ips;
int force_eom;
```

Output Fields

```
RES res;
```

Modified Fields

action, resolution, width, eff_page_type.

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.s_ips

Pointer to the infopkt stream designated for transmission.

args.force_eom

If set to 1, this value forces an EOM FSK message to be sent during the next page break instead of an MPS FSK.

Output

Return value:

1	An infopkt(s) indicating a beginning-of-page appeared and was processed.
0	End-of-file.
<0	Next infopkt does not indicate a beginning-of-page.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

This function sends the `end-of-page` between pages only. *BfvFaxEndOfDocument* sends the `end-of-page` after the last page has been sent.

BfvFaxNextPage peeks ahead in *args.s_ips*. If `end-of-file` is encountered, 0 is returned. If the next infopkt does not indicate a `beginning-of-page`, –1 is returned, and the infopkt is left at the head of the stream to be read by another function.

If the next infopkt indicates a `beginning-of-page`, 1 is returned, and that infopkt and all succeeding infopkts in *args.s_ips* that indicate a `beginning-of-page` are read and processed (for example, sending a new resolution to the channel).

If the next infopkt is not the first page of a document, an `end-of-page` is sent to the channel.

Infopkts that indicate a `beginning-of-page` are:

```
INFOPKT_BEGINNING_OF_PAGE
INFOPKT_DOCUMENT_PARAMETERS
INFOPKT_T30_PARAMETERS
INFOPKT_ASCII_PAGE_PARAMETERS
INFOPKT_PAGE_PARAMETERS
INFOPKT_FAX_HDR
INFOPKT_EFF_PAGE_PARAMETERS
```

This function is incorporated into the higher-level functions *BfvFaxPoll* and *BfvFaxSend*.

See Also

BfvFaxSendPage

Example

```
BTLINE *lp;
struct infopkt_stream *ips;
struct args_fax args;

for (;;)
{
    BT_ZERO(args);
    args.s_ips = ips;
    if (BfvFaxNextPage(lp, &args) <= 0)
        break;
    BT_ZERO(args);
    args.s_ips = ips;
    BfvFaxSendPage(lp, &args);
}
BfvFaxEndOfDocument(lp, &args);
```

BfvFaxNextPageDCX

Purpose

Sends an `end-of-page` in an Intel DCX fax file to the channel and processes parameters for the next page of data to be transmitted.

Syntax

```
int
BfvFaxNextPage DCX          (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
FILE *fp;
int combine;
int force_eom;
```

Output Fields

```
RES res;
```

Modified Fields

resolution, width.

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.fp

FILE * pointer returned by *fopen*.

args.combine

Indicates whether combination with earlier data will occur.

Valid values are:

0 Data not combined; page break.

1 Data combined; no page break.

`args.force_eom`

If set to 1, this value forces an EOM FSK message to be sent during the next page break instead of an MPS FSK.

Output

Return value:

1	The next page was read and its parameters processed.
0	End-of–file.
<0	An error condition occurred.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function sends the end-of-page to the channel between pages of a document only. *BfvFaxEndOfDocument* sends the `end-of-page` to the channel after the last page of a document is sent.

Use this function to transmit an Intel DCX fax file, which contains a set of Intel Bi-level PCX fax pages.

The *args.combine* argument will normally have a value of 0, causing it to send an `end-of-page` when appropriate. To force combination on a single page with data previously sent, the combine value must be set to 1. This value must also be set to 1 if the page begins with a function not designed for DCX files; for example, *BfvFaxBeginSendTiff* or *BfvFaxNextPageRaw*.

The application can call the *BfvFaxPageParams* function before calling this function with an *args.combine* value of 0.

After calling this function, the application can send other, non-DCX G3 or ASCII data for the channel to combine on the same page. The application would then call *BfvFaxNextPageDCX* with an *args.combine* value of 1 and *BfvFaxNextPageDCX* at the appropriate time. This sequencing permits the application to use the width and resolution from the DCX file for the page and for the strip containing the DCX image.

When ending a page and starting a new page, the application must call this function once with an *args.combine* value of 0 and an *args.fp* value for the next page. Therefore, if the function returns 0, indicating end-of-file, and the application intends to continue fax transmission with a new DCX file, the application must call this function again with an *args.combine* value of 0 and the new *args.fp* value before transmitting the new page.

To begin DCX transmission, the application must use one of the fax raw data functions, for example *BfvFaxBeginSendRaw*. You should specify the `RES_FINE` and `WIDTH_A4` when using DCX because all PCX pages are in this format.

Applications that create DCX files might find it useful to use the constant `DCX_ID` that the Dialogic® Brooktrout® header file defines.

See Also

BfvFaxSendPage

Example

```
BTLINE *lp;
FILE *fp;
struct args_fax *args;

for (;;)
{
    BT_ZERO(args);
    args.fp = fp;
    args.combine = 0;
    if (BfvFaxNextPageDCX(lp,&args) <= 0)
        break;
    BT_ZERO(args);
    args.fp = fp;
    BfvFaxSendPageDCX(lp,&args);
}
BfvFaxEndOfDocument(lp,&args);
```

BfvFaxNextPageRaw

Purpose

Sends an `end-of-page` from a raw data file to the channel and processes parameters for the next page of data to be transmitted.

Syntax

```
void
BfvFaxNextPageRaw          (lp, args)
   BTLINE                  *lp;
   struct args_fax         *args;
```

The structure contains the following fields.

Input Fields

```
int resolution;
int width;
int force_eom;
unsigned long eff_page_type;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.resolution

The resolution of the next page.

See the *BfvFaxBeginSendRaw* function for resolution values.

args.width

Horizontal width of the next page.

See the *BfvFaxBeginSendRaw* function for width values.

args.force_eom

If set to 1, this value forces an EOM FSK message to be sent during the next page break instead of an MPS FSK.

`args.eff_page_type`

Not supported on TruFax®.

If nonzero, the next transmitted page will be an enhanced fax format page of the specified type. Valid values are:

`BT_EFF_JPEG_ENABLE` 0x1
JPEG. May also logically OR in the following.

`BT_EFF_JPEG_FULLCOLOR` 0x2
Full color.

`BT_EFF_JPEG_DEFAULT_TABLES` 0x4
Default Huffman Tables.

`BT_EFF_JPEG_12BIT` 0x8
12 bits/pel, Otherwise 8.

`BT_EFF_JPEG_NOSUBSAMPLING` 0x10
No subsampling.

`BT_EFF_JPEG_CUSTOM_ILLUMINANT` 0x20
Custom Illuminant.

`BT_EFF_JPEG_CUSTOM_GAMUT` 0x40
Custom Gamut.

`BT_EFF_JBIG` 0x0100
JBIG. May also logically OR in the following.

`BT_EFF_JBIG_L0` 0x0200
L0 Mode.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function sends the `end-of-page` to the channel between pages of a document only.

BfvFaxEndOfDocument sends the `end-of-page` to the channel after the last page of a document is sent.

The *BfvFaxPageParams* function can be called before this function.

Many fax devices do not support the reception of enhanced fax format pages. To verify that the receiver can support the format you wish to send, you should use the T.30 holdup feature and examining the DIS (see *BfvFaxT30Holdup* and *LINE_DIS_DTC (lp)*).

See Also

BfvFaxSendData, *BfvFaxSendFile*

Example

```
BTLINE *lp;
struct args_fax args;

BT_ZERO(args);
args.fname = "page_one";
args.fmt = DATA_G3;
BfvFaxSendFile(lp, &args);
/* Begin second page, normal resolution, standard (A4)
   width */
args.resolution = RES_200H_100V;
args.width = WIDTH_A4;
BfvFaxNextPageRaw(lp, &args);
BT_ZERO(args);
args.fname = "page_two";
args.fmt = DATA_ASCII;
BfvFaxSendFile(lp, &args);
```

BfvFaxNextPageTiff

Purpose

Sends an `end-of-page` from a TIFF-F file to the channel and processes parameters for the next page of data to be transmitted.

Syntax

```
int
BfvFaxNextPageTiff          (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
TFILE *s_tp;
int combine;
int force_eom;
```

Output Fields

```
RES res;
```

Modified Fields

resolution, width.

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.s_tp

FILE * pointer returned by *fopen*.

args.combine

Indicates whether combination with earlier data will occur.

Valid values are:

0 Data not combined; page break.

1 Data combined; no page break.

`args.force_eom`

If set to 1, this value forces an EOM FSK message to be sent during the next page break instead of an MPS FSK.

Output

Return value:

1	The next page was read and its parameters processed.
0	End-of-file.
<0	An error condition occurred.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function sends the `end-of-page` to the channel between pages of a document only. *BfvFaxEndOfDocument* sends the `end-of-page` to the channel after the last page of a document is sent.

The *args.combine* argument will normally have a value of 0, causing it to send an `end-of-page` when appropriate. To force combination on a single page with data previously sent, the *args.combine* value must be set to 1. This value must also be set to 1 if the page began with a function not designed for TIFF files; for example, *BfvFaxBeginSendRaw*.

The application can call the *BfvFaxPageParams* function before calling this function with an *args.combine* value of 0.

After calling this function, the application can send other, non-TIFF G3 or ASCII data for the channel to combine on the same page. The application would then call *BfvFaxNextPageTiff* with an *args.combine* value of 1 and *BfvFaxSendPageTiff* at the appropriate time. This sequencing permits the application to use the width and resolution from the TIFF file for the page and for the strip containing the TIFF image.

When ending a page and starting a new page, the application must call this function once with an *args.combine* value of 0 and an *args.s_tp* value for the next page. Therefore, if the function returns 0, indicating `end-of-file`, and the application intends to continue fax transmission with a new TIFF file, the application must call this function again with an *args.combine* value of 0 and the new *args.s_tp* value before transmitting the new page.

TIFF-F, as supported by Dialogic® Brooktrout® boards, only supports MH, MR, and MMR. It does not support JPEG, JBIG, or any other enhanced fax formats.

See Also

BfvFaxSendPageTiff

Example

```
BTLINE *lp;
TFILE *tp;
struct args_fax args;

for (;;)
{
    BT_ZERO(args);
    args.tp = tp;
    args.combine = 0;
    if (BfvFaxNextPageTiff(lp, &args) <= 0)
        break;
    BT_ZERO(args);
    args.tp = tp;
    BfvFaxSendPageTiff(lp, &args);
}
BfvFaxEndOfDocument(lp, &args);
```

BfvFaxPageParams

Purpose

Sets the page parameters for subsequent pages of data when using raw data files.

Syntax

```
void
BfvFaxPageParams            (lp, args)
   BTLINE                   *lp;
   struct args_fax_page_params *args;
```

The structure contains the following fields.

Input Fields

```
unsigned top_margin;
unsigned bottom_margin;
unsigned length;
unsigned ascii_pad;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.top_margin

Specifies the top margin in units of tenths of an inch.

args.bottom_margin

Specifies the bottom margin in units of tenths of an inch.

args.length

Specifies the length of a page (normal resolution) in G3 lines. (97.79 normal resolution G3 lines/inch; 195.58 fine resolution G3 lines/inch).

args.ascii_pad

Specifies whether or not to pad short pages with blank scan lines to size *args.length*. Valid values are:

0 Pad.

1 Do not pad.

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

Call this function before beginning transmission with *BfvFaxBeginSendRaw* or *BfvFaxBeginRaw* or before beginning a new page with *BfvFaxNextPageRaw*.

Use the `pageparampkt` infopkt type to set these parameters when using infopkt-formatted data files. See *Volume 6, Appendix E, Infopkt Parameter Values* for more information on page parameters.

Example

```
struct args_fax args;

BT_ZERO(args);
arg.top_margin = 3;
args.bottom_margin = 3;
args.length = 1143;
args.ascii_pad = 0;
BfvFaxPageParams(lp, &args);
```

BfvFaxPoll

Purpose

Performs fax polling depending on the channel's mode, capability, and the results of the interrupts.

Syntax

```
void
BfvFaxPoll                  (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
int oa_flag;
struct infopkt_stream *s_ips;
struct infopkt_stream *r_ips;
char *local_id;
int xmit_mode;
int rcv_mode;
int btg3;
int force_eom;
int ecm_override;
```

Output Fields

```
FAX_RES fax_res;
RES res;
```

Modified Fields

can_send, can_receive, resolution, width, overlay_number, action, buf, size, fmt, expect_another, placement, insert_mode, label, spacing, units, s_tp, combine, fp, resolution_negot, use_open_file, open_file_size, fname, remote_info, btg3, eff_page_type.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.oa_flag

Sets the channel to dial or answer mode.

BT_ORIGINATEDial mode.

BT_ANSWERAnswer mode.

args.s_ips

Pointer to the infopkt stream ready for transmission.

NULL = Sending is disabled.

args.r_ips

Pointer to the infopkt stream ready for data reception.

NULL = Receiving is disabled.

args.local_id

Pointer to an ASCII string of digits and/or alphanumerics used as the TSI, CSI, or CIG ID string.

This ID string supersedes the ID string in the user-defined configuration file unless the value of *args.local_id* is NULL. In the case of NULL, the user-defined configuration file *id_string* remains unchanged.

This argument is ignored in countries that do not permit changes to the *local_id*.

args.xmit_mode

Specifies the fax transmit mode. Valid values are:

XMIT_MODE_AUTO Auto transmit mode; standard, default.	0
XMIT_MODE_MANUAL Manual transmit mode; same as AUTO, but has timing differences when retransmitting FSK signals.	1

`args.rcv_mode`

Specifies the fax receive mode. Valid values are:

`RCV_MODE_AUTO` 0
: Auto receive mode; standard, default.

`RCV_MODE_MANUAL` 1
: Manual receive mode; same as `AUTO`, but no initial CED is played and startup delays are reduced.

`args.btg3`

If set to 1, this value requests the function to interpret *btg3* headers. If a `BTG3` or `INDIR_BTG3` infopkt appears in the transmit infopkt stream, the resolution and width stored within the *btg3* header will be used as the strip resolution and width.

`args.force_eom`

If set to 1, this value forces an EOM FSK message to be sent during the next page break instead of an MPS FSK.

`args.ecm_override`

If set to 0, there is no effect; if > 0, ECM is enabled, overriding the setting of ecm_enable in the user configuration file; if < 0, ECM is disabled, overriding the settings of ecm_enable and t34_enable in the user configuration file. See the ecm_enable parameter in *Volume 6, Appendix A, User-defined Configuration File.*

Output

Return value: None.

`args.fax_res`

A structure containing information about the completed fax session. For a detailed description of the `FAX_RES` structure parameters, see *Volume 6, Result Structures, Appendix B*. See the `max_pagelist` parameter in *Volume 6, Appendix A*. The Bfv API automatically allocates and stores `PAGE_RES` structures in a linked list within *args.fax_res*. The application must free these structures after use to release the memory.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Not supported on TruFax®.

This is a high-level function to perform fax polling.

Performs Dial and Send, Dial and Receive, Answer and Send, Answer and Receive, Dial and Send and Receive, or Answer and Receive and Send fax, depending on the results of the `training_complete` and `turnaround` interrupts.

The *args.oa_flag* sets the channel to the Dial or Answer mode, the *args.s_ips* and *args.r_ips* arguments set the channel's capability to send or receive facsimiles or to perform both functions, and the results of the interrupts (`training_complete/turnaround`) trigger the actions that the channel performs.

The channel's capability is determined by the *args.s_ips* and *args.r_ips* arguments:

Channel Capability	*args.s_ips* set to	*args.r_ips* set to
Receive only	Null	Non-Null
Send only	Non-Null	Null
Send and receive	Non-Null	Non-Null
Error condition	Null	Null

If args.oa_flag = `ORIGINATE`

If received interrupt = `training_complete`

Go to `SEND`.

If received interrupt = `turnaround`

Go to `RECEIVE`.

If args.oa_flag = `ANSWER`

If received interrupt = `training_complete`

Go to `RECEIVE`.

If received interrupt = `turnaround`

Go to `SEND`.

`SEND`: *Send from infopkt stream args.s_ips.*

If received interrupt = `hangup`

`DONE`.

If received interrupt = `turnaround`

Go to `RECEIVE`.

RECEIVE: *Receive into infopkt stream args.r_ips.*

If received interrupt = hangup

DONE.

If received interrupt = turnaround

Go to SEND.

The first infopkt in *args.s_ips* must be:

INFOPKT_DOCUMENT_PARAMETERS

This function sets parameters based on the infopkts appearing at the start of *args.s_ips* that indicate a beginning-of-page.

Infopkts that indicate a beginning-of-page are:

INFOPKT_BEGINNING_OF_PAGE
INFOPKT_DOCUMENT_PARAMETERS
INFOPKT_T30_PARAMETERS
INFOPKT_ASCII_PAGE_PARAMETERS
INFOPKT_PAGE_PARAMETERS
INFOPKT_FAX_HDR
INFOPKT_EFF_PAGE_PARAMETERS

See Also

BfvFaxReceive, *BfvFaxSend*

Example

```
BTLINE *lp;
struct infopkt_stream *s_ips, *r_ips;
struct args_infopkt args_infopkt;
struct args_fax args_fax;

BT_ZERO(args_infopkt);
args_infopkt.fname = "sendfile";
args_infopkt.fmode = "r";
s_ips = BfvInfopktOpen(&args_infopkt);
args_infopkt.fname = "recfile";
args_infopkt.fmode = "w";
r_ips = BfvInfopktOpen(&args_infopkt);
/* If the remote fax machine permits, send a fax and, */
/* if the remote fax machine permits, receive a fax   */

BT_ZERO(args_fax);
args_fax.oa_flag = ORIGINATE;
args_fax.s_ips = s_ips;
args_fax.r_ips = r_ips;
args_fax.local_id = "local_id";
BfvFaxPoll(lp, &args_fax);
```

BfvFaxRcvPageDCX

Purpose

Receives a fax into an Intel DCX fax file that contains a set of Intel bi-level PCX fax pages.

Syntax

```
int
BfvFaxRcvPageDCX            (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

FILE *fp;

Output Fields

RES *res;

Modified Fields

fmt, resolution, width, expect_another, resolution_negot, force_res, btg3, buf, size, eff_page_type.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fp

FILE * pointer returned by fopen with mode "w+"/"wb+".

Output

Return value:

1	Another page is waiting to be transferred.
0	No more pages are waiting to be transferred.
<1	An error condition occurred.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

To use this function, the application must have opened the file using `fopen` with mode "w+"/"wb+".

An application uses this function to receive a fax into an Intel DCX fax file that contains a set of Intel bi-level PCX fax pages. The function can also transfer a page of G3 data from the driver buffer to a PCX page within the specified DCX file.

Prior to beginning DCX reception, the application must call *BfvFaxSetReceiveFmt* to set the format to `FMT_PCX_BILEVEL`.

After the last page is received and zero (0) is returned, call the *BfvFaxEndReception* function.

Applications that create DCX files might find it useful to use the constant `DCX_ID` that the header file defines.

Example

```
FILE *fp = fopen("test.dcx","w+");
struct args_fax args;

do
{
    BT_ZERO(args);
    args.fp = fp;
}
while (BfvFaxRcvPageDCX(lp, &args) > 0);
BfvFaxEndReception(lp, &args);
```

Note: When using Windows, it may be necessary to substitute "fopen" with "_dll_fopen" to avoid problems with differences in C runtime libraries.

BfvFaxRcvPageTiff

Purpose

Transfers a page of G3 data from the driver buffer to the specified TIFF file, along with appropriate TIFF tags and, optionally, a set of user-supplied TIFF tags.

Syntax

```
int
BfvFaxRcvPageTiff          (lp, args)
   BTLINE                  *lp;
   struct args_fax         *args;
```

The structure contains the following fields.

Input Fields

```
TFILE *r_tp;
struct ifd_field *opt_tags;
int num_opt_tags;
int compat_width;
```

Output Fields

```
RES res;
```

Modified Fields

fmt, buf, size, resolution, width, expect_another, resolution_negot, force_res, eff_page_type.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.r_tp

TFILE * pointer to a TIFF file.

args.opt_tags

Pointer to an array of structures of type *struct ifd_field* that contain extra tags to be written in addition to those normally created in a received fax document.

`args.num_opt_tags`

The number of tags specified in *args.opt_tags*. Can be 0.

`args.compat_width`

If nonzero, files produced for resolutions 300Hx300V and 400Hx400V will store width values readable by a previous Bfv API version 4.0.

Output

Return value:

1	Another page is waiting to transfer.
0	No more pages are waiting to transfer.
–1	An error condition occurred.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

After the last page is received and zero (0) is returned, call the *BfvFaxEndReception* function.

The *args.opt_tags* argument permits the user to optionally specify a set of IFD entries to be written to the TIFF-F file with the rest of the data for the received page. The user must know about TIFF file formats to make use of this feature.

The *args.opt_tags* argument contains a pointer to the first element of an array of *struct ifd_field* structures. Only IFD entries with field type `FT_BYTE`, `FT_SHORT`, or `FT_LONG`, with a count of 1, or with field type `FT_ASCII` are permitted. For field type FT_ASCII, set the offset to be the string pointer (cast to MILL_PTR_INT_TYPE).

All tags that are required for use with TIFF-F are automatically added to the IFD and should not be specified in *args.opt_tags*. These tags are:

TAG_NEWSUBFILETYPE	TAG_IMAGEWIDTH
TAG_IMAGELENGTH	TAG_BITSPERSAMPLE
TAG_COMPRESSION	TAG_PHOTOMETRICINTERP
TAG_FILLORDER	TAG_SAMPLESPERPIXEL
TAG_ROWSPERSTRIP	TAG_XRESOLUTION
TAG_YRESOLUTION	TAG_T4OPTIONS
TAG_T6OPTIONS	TAG_RESOLUTIONUNIT
TAG_PAGENUMBER	TAG_BADFAXLINES
TAG_CLEANFAXDATA	

In addition, the underlying function *BfvTiffWriteIFD* does not allow specification of TAG_STRIPOFFSETS or TAG_STRIPBYTECOUNTS.

The argument field *args.compat_width* can be used to produce files that, for high resolutions, are compatible with those that Bfv API version 4.0 produced and expected to read. These values were incorrect, and this option should only be used if such compatibility is required.

TIFF-F, as supported by Dialogic® Brooktrout® boards, only supports MH, MR, and MMR. It does not support JPEG, JBIG, or any other enhanced fax formats.

See Also

BfvFaxEndReception

Example

```
BTLINE *lp;
TFILE *tp;
struct args_fax args;

for (;;)
{
    BT_ZERO(args);
    args.r_tp = tp;
}
while (BfvFaxRcvPageTiff(lp, &args) > 0);
BfvFaxEndReception(lp, &args);
```

BfvFaxReceive

Purpose

Receives documents for storing as an infopkt stream.

Syntax

```
void
BfvFaxReceive                (lp, args)
   BTLINE                    *lp;
   struct args_fax           *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *r_ips;
char *local_id;
int rcv_mode;
unsigned fallback_rtp_reinvite;
int ecm_override;
int override_restrict_res;
int restrict_res;
int override_eff_pt_caps;
int eff_pt_caps;
int override_max_width;
int max_width;
int v34_override;
int override_line_compression;
int line_compression;
```

Output Fields

```
FAX_RES fax_res;
RES res;
```

Modified Fields

remote_info, buf, size, expect_another, resolution, width, resolution_negot, eff_page_type.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

`args.r_ips`

Pointer to the infopkt stream where the received data is stored.

`args.local_id`

Pointer to the ASCII string that contains the ID for transmission as part of the CSI.

This ID string supersedes the ID string in the user-defined configuration file unless the value of the *local_id* is `NULL`. In the case of `NULL`, the ID string remains unchanged.

This argument is ignored in countries that do not permit changes to the local ID.

`args.rcv_mode`

Specifies the fax receive mode. Valid values are:

`RCV_MODE_AUTO`	0
Auto receive mode; standard, default.	
`RCV_MODE_MANUAL`	1
Manual receive mode; same as `AUTO`, but no initial CED is played and startup delays are reduced.	

`args.fallback_rtp_reinvite`

Specifies whether or not a SIP RTP reINVITE should be transmitted for G.711 fallback mode if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable). Valid values are:

BT_FALLBACK_RTP_REINVITE_DEFAULT

Default to the setting specified in the call control configuration file.

BT_FALLBACK_RTP_REINVITE_DISABLE

Do not transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

BT_FALLBACK_RTP_REINVITE_ENABLE

Transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

When the application sets the value in this field to *BT_FALLBACK_RTP_REINVITE_DEFAULT*, the functionality will default to the setting specified in the call control configuration file (*callctrl.cfg*) by the *g711_fallback_rtp_reinvite* parameter.

If the *g711_fallback_rtp_reinvite* parameter isn't specified in the call control configuration file, then the default functionality is *BT_FALLBACK_RTP_REINVITE_DISABLE*.

An application can override the value specified in the call control configuration file by the *g711_fallback_rtp_reinvite* parameter by setting this field to a value of either *BT_FALLBACK_RTP_REINVITE_DISABLE* or *BT_FALLBACK_RTP_REINVITE_ENABLE*.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_DISABLE* will prevent transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected.

Setting this field to a value of *BT_FALLBACK_RTP_REINVITE_ENABLE* will result in transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable) and the fax transport protocol (*fax_transport_protocol*) parameter specified in the call control configuration file is set to *t38_first*. The SDP settings in the SIP RTP reINVITE will be the same RTP codec settings initially used to establish the call.

Note: This field only works for calls using the SIP internet protocol. The Bfv API ignores this field for calls using the H.323 internet protocol or PSTN line types.

`args.ecm_override`

If set to 0, there is no effect; if > 0, ECM is enabled, overriding the setting of ecm_enable in the user configuration file; if < 0, ECM is disabled, overriding the settings of ecm_enable and t34_enable in the user configuration file. See the ecm_enable parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_restrict_res`

If non-zero, args.restrict_res is active.

`args.restrict_res`

If args.override_restrict_res is non-zero, the specified value overrides the value of restrict_res specified in the user configuration file. See the restrict_res parameter in *Volume 6, Appendix A, User-defined Configuration File.*

`args.override_eff_pt_caps`

If non-zero, args.eff_pt_caps is active.

`args.eff_pt_caps`

If args.override_eff_pt_caps is non-zero, the specified value overrides the value of eff_pt_caps specified in the user configuration file. See the eff_pt_caps parameter in *Volume 6, Appendix A, User-defined Configuration File*.

`args.override_max_width`

If non-zero, args.max_width is active.

`args.max_width`

If args.override_max_width is non-zero, the specified value overrides the value of max_width specified in the user configuration file. See the max_width parameter in *Volume 6, Appendix A, User-defined Configuration File*.

`args.v34_override`

If set to 0, there is no effect; if > 0, V.34 is enabled, overriding the setting of v34_enable in the user configuration file; if < 0, V.34 is disabled, overriding the settings of v34_enable in the user configuration file. See the v34_enable parameter in *Volume 6, Appendix A, User-defined Configuration File*.

`args.override_line_compression`

If non-zero, args.line_compression is active.

`args.line_compression`

If args.override_line_compression is non-zero, the specified value overrides the value of line_compression specified in the user configuration file. See the line_compression parameter in *Volume 6, Appendix A, User-defined Configuration File*.

Output

`args.fax_res`

A structure containing information about the completed fax session. For a detailed description of the `FAX_RES` structure parameters, see *Volume 6, Appendix B*. See also the `max_pagelist` parameter in *Volume 6, Appendix A, Configuration Files, User-defined Configuration File*.

The Bfv API automatically allocates and stores `PAGE_RES` structures in a linked list within *args.fax_res*. The application must free these structures after use to release the memory.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This is a high-level function that:

- Receives fax pages and writes them to a file.
- Records the filenames and page parameters in an infopkt stream.

This function provides less flexibility than the low-level functions which implement it. For example, screening based on the remote ID and NSF/NSS information is not possible.

See Also

BfvFaxPoll, *BfvFaxSend*

Example

```
BTLINE *lp;
struct infopkt_stream *r_ips;
struct args_infopkt args_infopkt;
struct args_fax args_fax;

BT_ZERO(args_infopkt);
args_infopkt.fname = "recfile";
args_infopkt.fmode = "w";
r_ips = BfvInfopktOpen (&args_infopkt);
BT_ZERO(args_fax);
args_fax.r_ips = r_ips;
args_fax.local_id = "local_id";
BfvFaxReceive(lp, &args_fax);
```

BfvFaxReceiveData

Purpose

Receives raw fax data into a user-supplied buffer.

Syntax

```
int
BfvFaxReceiveData          (lp, args)
   BTLINE                  *lp;
   struct args_fax         *args;
```

The structure contains the following fields.

Input Fields

```
unsigned char *buf;
unsigned size;
```

Output Fields

```
int resolution;
int width;
int expect_another;
int resolution_negot;
unsigned long eff_page_type;
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.buf

Pointer to a user-allocated data buffer to hold the received data. The data buffer must be at least 1024 bytes in size.

args.size

Size, in bytes, of the user-allocated data buffer.

Output

Return value:

0	A full page was successfully transferred into the user buffer.
>0	The number of valid bytes successfully transferred into the user buffer. The number of bytes may be less than a full page.
<0	An error condition occurred.

`args.resolution`

The resolution of the incoming fax data. The value stored is only guaranteed to be valid on the first call to this function for a page. See the *BfvFaxBeginSendRaw* function for resolution values.

`args.width`

The width of the incoming fax. The value stored is only guaranteed to be valid on the first call to this function for a page. See the *BfvFaxBeginSendRaw* function for width values.

`args.expect_another`

A flag that indicates whether another page of incoming data exists. Values are:

0	No more pages exist.
1	Another page exists.

The value stored is only guaranteed to be valid when the return value from this function is 0 (indicating successful completion of a page).

`args.resolution_negot`

The negotiated incoming fax resolution. This value may differ from that of *resolution* if the *force_res* option of *BfvFaxSetReceiveFmt* is used.

`args.eff_page_type`

Not supported on TruFax®.

If nonzero, the current page is an enhanced fax format page of the specified type. Valid values are:

`BT_EFF_JPEG_ENABLE` 0x1
JPEG. May also logically OR in the following.

`BT_EFF_JPEG_FULLCOLOR` 0x2
Full color.

`BT_EFF_JPEG_DEFAULT_TABLES` 0x4
Default Huffman Tables.

`BT_EFF_JPEG_12BIT` 0x8
12 bits/pel, Otherwise 8.

`BT_EFF_JPEG_NOSUBSAMPLING` 0x10
No subsampling.

`BT_EFF_JPEG_CUSTOM_ILLUMINANT` 0x20
Custom Illuminant.

`BT_EFF_JPEG_CUSTOM_GAMUT` 0x40
Custom Gamut.

`BT_EFF_JBIG` 0x0100
JBIG. May also logically OR in the following.

`BT_EFF_JBIG_L0` 0x0200
L0 Mode.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function should be called repeatedly until 0 is returned or an error occurs.

Call the *BfvFaxEndReception* function after the last page is received.

If the application selects FMT_PCX_BILEVEL, You should avoid use of the *BfvFaxReceiveData* function to receive the fax because the PCX header value containing the page length will be incorrect.

On successful return, the PAGE_RES structure containing information about the received page is complete and can be examined. See *Volume 6, Appendix B, Bfv API Structures, PAGE_RES Structure Parameters.*

See Also

BfvFaxEndReception

Example

```
BTLINE *lp;
unsigned char buf[1024];
unsigned size;
struct args_fax args;

for (;;)
{
    BT_ZERO(args);
    args.buf = buf;
    args.size = sizeof(buf);
    size = BfvFaxReceiveData(lp,&args);
    if (size == 0)
        break;
    process_data(buf,size);
}
```

BfvFaxReceiveFile

Purpose

Receives a raw fax page to a file.

Syntax

```
void
BfvFaxReceiveFile          (lp, args)
   BTLINE                  *lp;
   struct args_fax         *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
int use_open_file;
FILE *fp;
int btg3;
```

Output Fields

```
int resolution;
int width;
int expect_another;
int resolution_negot;
unsigned long eff_page_type;
RES res;
```

Modified Fields

buf, size.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fname

Pointer to the data file in which to store the incoming data.

args.use_open_file

When set to 1, uses an opened file pointer supplied in *args.fp*.

`args.fp`

If *args.use_open_file* is set to 1, supplies an opened **`FILE`** * pointer.

If used in conjunction with PCX reception, the application must open the file it uses in “w+”/”wb+” mode.

`args.btg3`

If set to 1, this value requests the function to create a *btg3* header. The resolution and width will be stored within the header.

Output

Return value: None.

`args.resolution`

The resolution of the incoming fax data. See the *BfvFaxBeginSendRaw* function for resolution values.

`args.width`

The width of the incoming fax. See the *BfvFaxBeginSendRaw* function for width values.

`args.expect_another`

A flag that indicates whether another page of incoming data exists. Values are:

0 No more pages exist.

1 Another page exists.

`args.resolution_negot`

The negotiated incoming fax resolution. This value may differ from that of *args.resolution* if the *force_res* option of *BfvFaxSetReceiveFmt* is used.

`args.eff_page_type`

Not supported on TruFax®.

If nonzero, the current page is an enhanced fax format page of the specified type. Valid values are:

`BT_EFF_JPEG_ENABLE` 0x1
JPEG. May also logically OR in the following.

`BT_EFF_JPEG_FULLCOLOR` 0x2
Full color.

`BT_EFF_JPEG_DEFAULT_TABLES` 0x4
Default Huffman Tables.

`BT_EFF_JPEG_12BIT` 0x8
12 bits/pel, Otherwise 8.

`BT_EFF_JPEG_NOSUBSAMPLING` 0x10
No subsampling.

`BT_EFF_JPEG_CUSTOM_ILLUMINANT` 0x20
Custom Illuminant.

`BT_EFF_JPEG_CUSTOM_GAMUT` 0x40
Custom Gamut.

`BT_EFF_JBIG` 0x0100
JBIG. May also logically OR in the following.

`BT_EFF_JBIG_L0` 0x0200
L0 Mode.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Call *BfvFaxEndReception* after the last page is received.

On successful return, the `PAGE_RES` structure containing information about the received page is complete and can be examined. See *Volume 6, Appendix B, Bfv API Structures, PAGE_RES Structure Parameters*.

See Also

BfvFaxEndReception, `LINE_SET_FILE_OPEN`

Example

```
BTLINE *lp;
struct args_fax args;

BT_ZERO(args);
args.fname = "fax1";
BfvFaxReceiveFile(lp, &args);
```

BfvFaxReceivePage

Purpose

Receives a fax page to an infopkt stream.

Syntax

```
int
BfvFaxReceivePage          (lp, args)
   BTLINE                  *lp;
   struct args_fax         *args;
```

The structure contains the following fields.

Input Fields

struct infopkt_stream **r_ips;*

Output Fields

RES *res;*

Modified Fields

buf, size, expect_another, resolution, width, resolution_negot, eff_page_type.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.r_ips

Pointer to the infopkt stream where the received data is stored.

Output

Return value:

1	Another page is waiting to transfer.
0	No more pages are waiting to transfer.
<0	An error condition occurred.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

After the last page is received and zero is returned, call the *BfvFaxEndReception* function.

This function is incorporated into the higher-level function *BfvFaxReceivePages*.

On successful return, the PAGE_RES structure containing information about the received page is complete and can be examined. See *Volume 6, Appendix B, Bfv API Structures, PAGE_RES Structure Parameters*.

See Also

BfvFaxReceivePages, *BfvFaxEndReception*

Example

```
BTLINE *lp;
struct infopkt_stream *ips;
struct args_fax args;

for (;;)
{
    BT_ZERO(args);
    args.r_ips = ips;
}
while (BfvFaxReceivePage(lp, &args) > 0);
BfvFaxEndReception(lp, &args);
```

BfvFaxReceivePages

Purpose

Receives multiple pages of fax data to an infopkt stream.

Syntax

```
void
BfvFaxReceivePages          (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

struct infopkt_stream **r_ips;*

Output Fields

RES *res;*

Modified Fields

buf, size, expect_another, resolution, width, resolution_negot, eff_page_type.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.r_ips

Pointer to the infopkt stream where the received data is stored.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

On return from this function, the last page was received from the remote fax machine and:

- Either the channel went on-hook, and the line state was changed to `IDLE`.

or

- The channel remained off-hook, and the line state was changed to `TURNAROUND` if polling was enabled using *BfvFaxBegin* or *BfvFaxBeginSendRaw*. The receiver then becomes a transmitter, and *BfvFaxGetRemoteInfo* and *BfvFaxWaitForTraining* must be called again.

This function is incorporated into the higher-level functions *BfvFaxPoll* and *BfvFaxReceive*.

See Also

BfvFaxReceivePage

Example

```
BTLINE *lp;
struct infopkt_stream *ips;
struct args_fax args;

BT_ZERO(args);
args.r_ips = ips;
BfvFaxReceivePages(lp, &args);
```

BfvFaxSend

Purpose

Sends documents as an infopkt stream.

Syntax

```
void
BfvFaxSend                  (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *s_ips;
char *local_id;
int xmit_mode;
int btg3;
int force_eom;
int ecm_override;
```

Output Fields

```
FAX_RES fax_res;
RES res;
```

Modified Fields

resolution, width, remote_info, overlay_number, action, placement, insert_mode, label, spacing, units, buf, size, fmt, s_tp, combine, fp, open_file_size, fname, btg3, eff_page_type.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.s_ips

Pointer to the infopkt stream ready for transmission.

`args.local_id`

Pointer to the ASCII string of digits and/or alphanumerics used as the TSI ID string. This ID string supersedes the ID string in the user-defined configuration file unless the value of the *local_id* is `NULL`. In the case of `NULL`, the ID string remains unchanged.

This argument is ignored in countries that do not permit changes to the local ID.

`args.xmit_mode`

Specifies the fax transmit mode. Valid values are:

`XMIT_MODE_AUTO` 0

Auto transmit mode; standard, default.

`XMIT_MODE_MANUAL` 1

Manual transmit mode; same as `AUTO`, but has timing differences when retransmitting FSK signals.

`args.btg3`

If set to 1, this value requests the function to interpret *btg3* headers. If a `BTG3` or `INDIR_BTG3` infopkt appears in the transmit infopkt stream, the resolution and width stored within the *btg3* header will be used as the strip resolution and width.

If set to –1, a header appearing in the file will be skipped.

`args.force_eom`

If set to 1, this value forces an EOM FSK message to be sent during the next page break instead of an MPS FSK.

`args.ecm_override`

If set to 0, there is no effect; if > 0, ECM is enabled, overriding the setting of ecm_enable in the user configuration file; if < 0, ECM is disabled, overriding the settings of ecm_enable and t34_enable in the user configuration file. See the ecm_enable parameter in *Volume 6, Appendix A, User-defined Configuration File.*

Output

Return value: None.

`args.fax_res`

A structure containing information about the completed fax session. For a detailed description of the `FAX_RES` structure parameters, see *Volume 6, Appendix B, FAX_RES Structure Parameters*. See also the `max_pagelist` parameter in *Volume 6, Appendix A, User-defined Configuration File.*

The Bfv API automatically allocates and stores `PAGE_RES` structures in a linked list within *args.fax_res*. The application must free these structures after use to release the memory.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This is a high-level function that sends documents based on an infopkt stream.

The function provides less flexibility than the low-level functions that implement it. Screening transmissions based on the remote fax ID, poll for a fax document, or NSF/NSS (Non Standard Facility/Non Standard Set-up) information is not possible with this function.

Use the function after *BfvLineOriginateCall* to perform all steps necessary to transmit a fax and go off-hook.

The function sets parameters that indicate a `beginning-of-page` based on the infopkts appearing at the start of *args.s_ips*. Infopkts that indicate a `beginning-of-page` are:

```
INFOPKT_BEGINNING_OF_PAGE
INFOPKT_DOCUMENT_PARAMETERS
INFOPKT_T30_PARAMETERS
INFOPKT_ASCII_PAGE_PARAMETERS
INFOPKT_PAGE_PARAMETERS
INFOPKT_FAX_HDR
INFOPKT_EFF_PAGE_PARAMETERS
```

At function start, checks to ensure that the line state is set to `CONNECTED`. At function end, loops until fax transmission completes successfully and the line state is set to `IDLE`.

See Also

BfvFaxPoll, *BfvFaxReceive*

Example

```
BTLINE *lp;
struct infopkt_stream *s_ips;
struct args_infopkt args_infopkt;
struct args_fax args_fax;

BT_ZERO(args_infopkt);
args_infopkt.fname = “sendfile”;
args_infopkt.fmode = “r”;
s_ips = BfvInfopktOpen (&args_infopkt);
BT_ZERO(args_fax);
args_fax.s_ips = s_ips;
args_fax.local_id = “local_id”;
BfvFaxSend(lp, &args_fax);
```

BfvFaxSendData

Purpose

Sends raw fax data from a user-supplied buffer.

Syntax

```
void
BfvFaxSendData              (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
unsigned char *buf;
unsigned size;
unsigned fmt;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.buf

Pointer to a user-allocated data buffer.

args.size

Size, in bytes, of the user-allocated data buffer.

args.fmt

Specifies the format of the data to be transmitted. Typical values are DATA_G3 and DATA_ASCII. See the *BfvFaxStripParams* and *BfvFaxSetReceiveFmt* functions for more information on data formats.

Output

Return value: None.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures*, in this document.

Details

Call the *BfvFaxNextPageRaw* or *BfvFaxEndOfDocument* function after this function. The *BfvFaxStripParams* function can be called before or after this function.

See Also

BfvFaxNextPageRaw, *BfvFaxSendFile*

Example

```
BTLINE *lp;
unsigned char buf[1024];
unsigned size;
struct args_fax args;

for (;;)
{
    size = read_data_from_file(buf);
    if (size == 0)
        break;
    BT_ZERO(args);
    args.buf = buf;
    args.size = size;
    BfvFaxSendData(lp,&args);
}
```

BfvFaxSendFile

Purpose

Sends a raw fax page from a file.

Syntax

```
void
BfvFaxSendFile              (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
unsigned fmt;
int use_open_file;
FILE *fp;
long open_file_size;
int btg3;
```

Output Fields

```
RES res;
```

Modified Fields

buf, size.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fname

Name of the file where the data to transmit is stored.

args.fmt

Specifies the format of the data to be transmitted. Typical values are DATA_G3 and DATA_ASCII. See the *BfvFaxStripParams* and *BfvFaxSetReceiveFmt* functions for more information on data formats.

`args.use_open_file`

When set to 1, uses an opened file pointer supplied in *args.fp* and a size supplied in *args.open_file_size*.

`args.fp`

If *args.use_open_file* is set to 1, supplies an opened **`FILE`** * pointer.

`args.open_file_size`

If *args.use_open_file* is set to 1, then if this value is nonzero it specifies a maximum number of bytes to send from the opened file.

`args.btg3`

If set to 1, this value requests the function to interpret *btg3* headers. If a header appears in the file, the resolution and width stored within will be used as the strip resolution and width.

If set to –1, a header appearing in the file will be skipped.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Call the *BfvFaxNextPageRaw* or *BfvFaxEndOfDocument* function after this function. The *BfvFaxStripParams* function can be called before this function.

If you call *BfvFaxSendFile* several times in a row without specifying a page break (*BfvFaxNextPageRaw*) in between, the function sends one large page and not multiple pages.

Upon successful return, fax data is sent to the board but not yet sent over the line. Therefore, the `PAGE_RES` structure is not complete. Use the `LINE_SET_PAGE_COMPLETE_FUNC` macro to determine completeness.

See Also

BfvFaxNextPageRaw, *BfvFaxSendData*, `LINE_SET_PAGE_COMPLETE_FUNC`

Example

```
BTLINE *lp;
struct args_fax args;

BT_ZERO(args);
args.fname = "page_one";
args.fmt = DATA_G3;
BfvFaxSendFile(lp, &args);
/* Begin second page, normal resolution, standard (A4)
   width */
args.resolution = RES_200H_100V;
args.width = WIDTH_A4;
BfvFaxNextPageRaw(lp, &args);
BT_ZERO(args);
args.fname = "page_two";
args.fmt = DATA_ASCII;
BfvFaxSendFile(lp, &args);
```

BfvFaxSendPage

Purpose

Sends one fax page from the infopkt stream and looks for an `end-of-file` or new page type infopkt before returning.

Syntax

```
int
BfvFaxSendPage              (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
struct infopkt_stream *s_ips;
int btg3;
int force_eom;
```

Output Fields

```
RES res;
```

Modified Fields

placement, insert_mode, label, spacing, units, buf, size, fmt, s_tp, combine, fp, resolution, width, open_file_size, fname, eff_page_type.

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

args.s_ips

Pointer to the infopkt stream designated for transmission.

args.btg3

If set to 1, this value requests the function to interpret *btg3* headers. If a `BTG3` or `INDIR_BTG3` infopkt appears in the transmit infopkt stream, the resolution and width stored within the *btg3* header will be used as the strip resolution and width.

args.force_eom

If set to 1, this value forces an EOM FSK message to be sent during the next page break instead of an MPS FSK.

Output

Return value:

0	An entire page was successfully transmitted to the driver buffer.
<0	Failed to transmit an entire page.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

A page can be composed of several strips that are combined by the channel as they are transmitted. The user must keep track of the results.

The function processes strip infopkts if they are first in the infopkt stream.

The function checks to ensure that the line state is set to FAX_MODE both at function start and in the loop that loads data into the driver buffer.

This function is incorporated into the higher-level functions *BfvFaxPoll* and *BfvFaxSend*.

Upon successful return, fax data is sent to the board but not yet sent over the line. Therefore, the PAGE_RES structure is not complete. Use the LINE_SET_PAGE_COMPLETE_FUNC macro to determine completeness.

See Also

BfvFaxNextPage, LINE_SET_PAGE_COMPLETE_FUNC

Example

```
BTLINE *lp;
struct infopkt_stream *ips;
struct args_fax args;

for (;;)
{
    BT_ZERO(args);
    args.s_ips = ips;
    if (BfvFaxNextPage(lp, &args) <= 0)
        break;
    BT_ZERO(args);
    args.s_ips = ips;
    BfvFaxSendPage(lp, &args);
}
BfvFaxEndOfDocument(lp, &args);
```

BfvFaxSendPageDCX

Purpose

Transmits one fax page from an Intel DCX fax file that contains a set of Intel bi-level PCX fax pages.

Syntax

```
int
BfvFaxSendPageDCX          (lp, args)
   BTLINE                  *lp;
   struct args_fax         *args;
```

The structure contains the following fields.

Input Fields

FILE *fp;

Output Fields

RES *res;*

Modified Fields

fmt, open_file_size, btg3, buf, size.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fp

FILE * pointer returned by *fopen*.

Output

Return value:

0	An entire page was successfully transmitted to the driver buffer.
<0	Transmission failed.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Call the *BfvFaxNextPageDCX* or *BfvFaxEndOfDocument* function after this function.

See Also

BfvFaxNextPageDCX

Example

```
BTLINE *lp;
FILE *fp;
struct args_fax args;

for (;;)
{
    BT_ZERO(args);
    args.fp = fp;
    args.combine = 0;
    if (BfvFaxNextPageDCX(lp, &args) <= 0)
        break;
    BT_ZERO(args);
    args.fp = fp;
    BfvFaxSendPageDCX(lp, &args);
}
BfvFaxEndOfDocument(lp, &args);
```

BfvFaxSendPageTiff

Purpose

Transmits one fax page from the TIFF-F file.

Syntax

```
int
BfvFaxSendPageTiff          (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

TFILE *s_tp;

Output Fields

RES *res;*

Modified Fields

buf, size, fmt.

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.s_tp

Pointer to a TIFF file.

Output

Return value:

0	An entire page was successfully transmitted to the driver buffer.
<0	Transmission failed.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Call the *BfvFaxNextPageTiff* or *BfvFaxEndOfDocument* function after this function.

TIFF-F, as supported by Dialogic® Brooktrout® boards, only supports MH, MR, and MMR. It does not support JPEG, JBIG, or any other enhanced fax formats.

See Also

BfvFaxNextPageTiff

Example

```
BTLINE *lp;
TFILE *tp;
struct args_fax args;

for (;;)
{
    BT_ZERO(args);
    args.tp = tp;
    args.combine = 0;
    if (BfvFaxNextPageTiff(lp, &args) <= 0)
        break;
    BT_ZERO(args);
    args.tp = tp;
    BfvFaxSendPageTiff(lp, &args);
}
BfvFaxEndOfDocument(lp, &args);
```

BfvFaxSetLocalId

Purpose

Sets the local ID to the specified string using only the first 20 characters of the string.

Syntax

```
int
BfvFaxSetLocalId            (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
char *local_id;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.local_id

A null terminated ASCII string that contains the local ID (usually the phone number) used in the TSI, CSI, or CIG commands. The maximum length of the string is 20 characters.

Output

Return value:

0	The local ID was successfully set.
<0	Failed to set the local ID.

Setting the local ID is not permitted in some countries.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

Use this command before initiating fax transmission or reception protocols (for example, using the *BfvFaxBeginSend* function).

The loaded value remains in effect until the channel is reset (*BfvLineReset*) or this function is re-executed.

This function is ignored in countries that do not permit changes to the local ID.

This function is incorporated into the higher-level functions *BfvFaxSend*, *BfvFaxReceive* and *BfvFaxPoll*.

See Also

BfvFaxSetNSF, *BfvFaxSetSubPwdSep*, *id_string* parameter in *Volume 6, Appendix A, Configuration Files, User-defined Configuration File.*

Example

```
BTLINE *lp;
struct args_fax args;

BT_ZERO(args);
args.local_id = "My_Id_String";
if (BfvFaxSetLocalId(lp, &args) < 0)
{
    printf("Unable to Change Local Id ");
    printf("Due to Country Restrictions.\n");
}
```

BfvFaxSetNSF

Purpose

Sets up an NSF/NSC or NSS FSK message for transmission to the remote host.

Syntax

```
void
BfvFaxSetNSF                    (lp, args)
   BTLINE                       *lp;
   struct args_fax              *args;
```

The structure contains the following fields.

Input Fields

```
int nsf_or_nss;
unsigned char *buf;
unsigned size;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.nsf_or_nss

Determines whether the NSF/NSC or NSS is set. Valid values are:

SET_NSF_NSC1

SET_NSS2

`args.buf`

Pointer to a user-supplied buffer that contains the message to set up for NSF/NSC or NSS transmission. For example:

Octal representation	`\001\002\003\004`
Hex representation	`\x01\x02\x03\x04`

`args.size`

The number of bytes contained in *args.buf*.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The meanings of NSF, NSS, and NSC messages are often defined by a particular fax machine manufacturer and are meaningful only to other such machines.

The message passed to this function should only include the contents of the message and omit any preliminary bytes or trailing CRC bytes.

Note: The firmware treats the BfvFaxSetNSF message as an ASCII null-terminated string. Therefore, the *BfvFaxSetNSF* function does not support message strings that contain an embedded byte value of 0 (NULL byte).

If enabled, the NSF is sent along with the DIS from receiver to transmitter; the NSS is sent along with the DCS from transmitter to receiver; the NSC is sent along with the DTC from receiver to transmitter.

You should verify that all applications that call this function use the T.30 holdup feature. Call this function to set the NSF/NSC before beginning fax transmission or reception and to set the NSS after entering the `T30H_SECTION_XMIT_DCS` holdup (see the *BfvFaxT30Holdup* function).

See Also

BfvFaxSetLocalId, *BfvFaxSetSubPwdSep*

Example

```
struct args_fax args;

BT_ZERO(args);
args.nsf_or_nss = SET_NSF_NSC;
args.buf = "\001\002\003\004";
args.size = 4;
BfvFaxSetNSF(lp, &args);
BfvFaxBeginReceive(lp, &args);
```

BfvFaxSetReceiveFmt

Purpose

Sets the format used to pass received fax data from the channel to the computer.

Syntax

```
void
BfvFaxSetReceiveFmt          (lp, args)
   BTLINE                    *lp;
   struct args_fax           *args;
```

The structure contains the following fields.

Input Fields

```
unsigned fmt;
int force_res;
int resolution;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fmt

FMT_MMR_UNALIGN_MSB, FMT_MMR_UNALIGN_LSB, FMT_MMR_ALIGN_MSB, and FMT_MMR_ALIGN_LSB are not supported on TruFax®.

Value indicating the format to use. Valid values are:

FMT_MH_UNALIGN_MSB 0x0

MH data, EOLs not byte-aligned, MSB order.

FMT_MH_UNALIGN_LSB 0x1

MH data, EOLs not byte-aligned, LSB order.

FMT_MH_ALIGN_MSB 0x2

MH data, EOLs byte-aligned, MSB order.

`FMT_MH_ALIGN_LSB` 0x3

MH data, EOLs byte-aligned, LSB order.

`FMT_MR_UNALIGN_MSB` 0x4

MR data, EOLs not byte-aligned, MSB order.

`FMT_MR_UNALIGN_LSB` 0x5

MR data, EOLs not byte-aligned, LSB order.

`FMT_MR_ALIGN_MSB` 0x6

MR data, EOLs byte-aligned, MSB order.

`FMT_MR_ALIGN_LSB` 0x7

MR data, EOLs byte-aligned, LSB order.

`FMT_PCX_BILEVEL` 0x9

Intel Bi-level PCX data.

`FMT_MMR_UNALIGN_MSB` 0x10

MMR data, EOLs not byte-aligned, MSB order.

`FMT_MMR_UNALIGN_LSB` 0x11

MMR data, EOLs not byte-aligned, LSB order.

`FMT_MMR_ALIGN_MSB` 0x12

MMR data, EOLs byte-aligned, MSB order.

`FMT_MMR_ALIGN_LSB` 0x13

MMR data, EOLs byte-aligned, LSB order.

`args.force_res`

If set to 1, enables the *args.resolution* field.

`args.resolution`

If *args.force_res* is set to 1, this value specifies the resolution of the received data to be supplied to the host.

See the *BfvFaxBeginSendRaw* function for resolution values.

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Applications can call this function before calling:

BfvFaxBeginReceive
BfvFaxReceive
BfvFaxBegin
BfvFaxBeginRaw
BfvFaxBeginTiff

To transmit data previously received in a nonstandard format, see the `data_fmt` field of the `struct g3strippkt` type in *Volume 6, Appendix E, Infopkt Parameter Values, ASCII Strip Infopkt*.

The received format remains in effect until the application calls this function again, or it resets the channel. If infopkt streams are used, the receive file contains a G3 strip parameter packet with the appropriate data format information in it. If noninfopkt files are used, applications must keep track of the data format.

If the application selects `FMT_PCX_BILEVEL`, You should avoid use of the *BfvFaxReceiveData* function to receive the fax because the PCX header value containing the page length will be incorrect.

The default format is `FMT_MH_UNALIGN_MSB`.

See Also

BfvFaxSetLocalId, *BfvFaxSetSubPwdSep*.

Example

```
BTLINE *lp;
struct args_fax args;

BT_ZERO(args);
args.fmt = FMT_MH_ALIGN_MSB;
BfvFaxSetReceiveFmt(lp, &args);
BfvFaxBeginReceive(lp, &args);
```

BfvFaxSetSubPwdSep

Purpose

Sets up a SUB, PWD, or SEP (FSK) message to send to the remote host.

Syntax

```
void
BfvFaxSetSubPwdSep          (lp, args)
   BTLINE                   *lp;
   struct args_fax          *args;
```

The structure contains the following fields.

Input Fields

```
unsigned selector;
unsigned char *buf;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.buf

Not supported on TruFax®.

Pointer to a user-supplied buffer that contains the message. The message must be 0-terminated.

`args.selector`

The FSK value to enable:

`FSK_SUB_1`

`FSK_PWD_XMIT_1`

`FSK_PWD_POLL`

`FSK_SEP`

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The message passed to this function should only include the contents of the message and omit any preliminary bytes or trailing CRC bytes.

If enabled, and if the channel is the transmitter, the channel sends the SUB or PWD message with the DCS to the receiver. If the channel becomes the receiver, it sends the PWD or SEP with the DTC to the transmitter. The channel sends the message only if the answering side sends a DIS that indicates it is capable of receiving the message (see the `subpwdsep` parameter in *Volume 6, Appendix A, User-defined Configuration File*).

The length of the user-supplied buffer, excluding the 0-termination, must be $\leq$ 20. If the length is < 20, the function pads the message on the left with spaces.

You should verify that all applications calling this function use the T.30 Holdup feature. Call this function to set the SUB message after entering the `T30H_SECTION_XMIT_DCS` holdup (see the *BfvFaxT30Holdup* function).

See Also

BfvFaxSetLocalId, *BfvFaxSetNSF*, *subpwdsep* parameter in *Volume 6, Appendix A, User-defined Configuration File.*

Example

```
struct args_fax args;

BT_ZERO(args);
args.selector = FSK_SUB_1;
args.buf = "123456";
BfvFaxSetSubPwdSep(lp, &args);
BfvFaxBeginSend(lp, &args);
```

BfvFaxStripParams

Purpose

Sets strip parameters and separates different data strips for raw data files.

Syntax

```
void
BfvFaxStripParams          (lp, args)
   BTLINE                  *lp;
   struct args_fax_strip_params *args;
```

The structure contains the following fields.

Input Fields

```
unsigned fmt;
int resolution;
unsigned width;
unsigned left_margin;
unsigned right_margin;
unsigned line_spacing;
unsigned eof_char;
unsigned font_no;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.fmt

Specifies the format of the data (ASCII or G3) and, when G3 data is specified, identifies the type of G3.

The values most commonly used are:

DATA_G3(= FMT_MH_UNALIGN_MSB)

DATA_ASCII(= FMT_ASCII)

Other valid values are:

FMT_MH_UNALIGN_MSB	0x0
FMT_MH_UNALIGN_LSB	0x1
FMT_MH_ALIGN_MSB	0x2
FMT_MH_ALIGN_LSB	0x3
FMT_MR_UNALIGN_MSB	0x4
FMT_MR_UNALIGN_LSB	0x5
FMT_MR_ALIGN_MSB	0x6
FMT_MR_ALIGN_LSB	0x7
FMT_ASCII	0x8
FMT_PCX_BILEVEL	0x9
FMT_MMR_UNALIGN_MSB	0x10
FMT_MMR_UNALIGN_LSB	0x11
FMT_MMR_ALIGN_MSB	0x12
FMT_MMR_ALIGN_LSB	0x13

See also the *BfvFaxSetReceiveFmt* function.

args.resolution

Specifies the resolution of the strip, which can differ from the resolution of the page (as specified in *BfvFaxBeginSendRaw*, *BfvFaxNextPageRaw*, or *BfvFaxBeginRaw*) containing the strip.

See the *BfvFaxBeginSendRaw* function for resolution values. ASCII strips are always in normal resolution.

args.width

Specifies the width of the strip, which can differ from the width of the page (as specified in *BfvFaxBeginSendRaw*, *BfvFaxNextPageRaw*, or *BfvFaxBeginRaw*) containing the strip.

See the *BfvFaxBeginSendRaw* function for width values.

args.left_margin

Specifies the left margin in units of tenths of an inch (ASCII only – Range of 0-12 units).

`args.right_margin`

Specifies the right margin in units of tenths of an inch (ASCII only – Range of 0-12 units).

`args.line_spacing`

Specifies the spacing between text lines in G3 lines (ASCII only).

`args.eof_char`

Specifies the ASCII end-of-file character (ASCII only).

`args.font_no`

Specifies the font to use (ASCII only). For more information, see the *font_file* parameter in *Volume 6, Appendix A, User-defined Configuration File* or *BfvFaxDownloadFont*. The range is 0-6. If the specified font has not been downloaded, a default font is used (see *BfvFaxDownloadFont* or the *font_file* parameter in *Volume 6, Appendix A*).

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

`FMT_MMR_UNALIGN_MSB`, `FMT_MMR_UNALIGN_LSB`, `FMT_MMR_ALIGN_MSB`, and `FMT_MMR_ALIGN_LSB` are not supported on TruFax®.

This function is normally called before *BfvFaxSendFile* or *BfvFaxSendData*. The *args.fmt* value in the subsequent calls to *BfvSendFile* or *BfvSendData* must match the value supplied to this function.

Use either the `asciistrippkt` or `g3strippkt` infopkt type to send strip parameters when using infopkt-formatted data files.

See *Volume 6, Appendix E, Infopkt Parameter Values* for more information on strip parameters and how they are used.

See Also

BfvFaxSetReceiveFmt

Example

```
struct args_fax_strip_params args;

BT_ZERO(args);
args.fmt = DATA_G3;
args.resolution = RES_200H_100V;
args.width = WIDTH_A4;
BfvFaxStripParams(lp, &args);

or

BT_ZERO(args);
args.fmt = DATA_ASCII;
args.resolution = RES_200H_100V;
args.width = WIDTH_A4;
args.left_margin = 5;
args.right_margin = 0;
args.line_spacing = 2;
args.eof_char = 0x1a;
args.font_no = 0;
BfvFaxStripParams(lp, &args);
```

BfvFaxT30Holdup

Purpose

Enables the T.30 holdup feature that causes the channel to wait during T.30 negotiations until told to continue, and calls the specified user-supplied function when a T.30 holdup does occur.

Syntax

```
void
BfvFaxT30Holdup              (lp, args)
   BTLINE                    *lp;
   struct args_fax           *args;
```

The structure contains the following fields.

Input Fields

```
int holdup_mode;
int (*func) (BTLINE *lp, char *arg, int section);
char *arg;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.holdup_mode

Specifies how T.30 holdup will operate. Logically OR together the following values:

T30_HOLDUP_NONE — 0

Disable T.30 holdup.

T30_HOLDUP_DCS — 1

Holds up on transmit before DCS transmission or on receive after DCS reception.

T30_HOLDUP_RCV_MCF — 2

Holds up on receive before page confirmation.

`args.func`

A pointer to a user-supplied integer function that is called when the channel enters a T.30 holdup state. Within this function, the application calls other functions that set T.30 parameters.

Args.func will be called as `(*args.func)(lp,args.arg,section)`.

The *lp* variable contains the pointer to the line structure; *args.arg* contains the supplied user-defined argument; *section* contains a value that indicates where in the T.30 protocol the holdup occurred. Valid values for *section* are:

`T30H_SECTION_XMIT_DCS`

`T30H_SECTION_RCV_DCS`

`T30H_SECTION_RCV_MCF`

Returns an integer:

0 Continues with fax procedure.

1 Aborts fax procedure.

`args.arg`

Argument to *args.func*. Can be `NULL`.

Output

Return value: None.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Applications can use this function in conjunction with any fax sending and/or receiving functions except the highest level infopkt functions *BfvFaxSend*, *BfvFaxReceive*, and *BfvFaxPoll*. When the channel has entered a T.30 holdup state and the user-supplied function is called, the Bfv API sets the line state to `LINE_STATE_HOLDUP`.

Using this feature enables an application and associated channel to delay making decisions about T.30 parameters until after they have received as much information as possible from the remote side.

A basic understanding of T.30 negotiations is helpful in understanding how to use this function. The information that follows applies to `T30_HOLDUP_DCS` holdup mode only.

The following scenario is a typical T.30 negotiation in which polling is not involved. When an answering fax machine (receiver) answers, it transmits a DIS signal, a CSI signal (optional), and one or more NSF (optional) signals. The calling fax machine (transmitter) responds with a DCS signal, a TSI signal (optional), and one or more NSS (optional) signals. The receiver then transmits a CFR signal.

In this scenario, the transmitter will enter a T.30 holdup state after receiving the DIS signal from the receiver, and the user-supplied function will be called with *section* set to `T30H_SECTION_XMIT_DCS`. The receiver will enter a T.30 holdup state after receiving the DCS signal from the transmitter, and the user-supplied function will be called with *section* set to `T30H_SECTION_RCV_DCS`.

If polling was involved at the start, the signaling sequence would be DIS (CSI NSF), DTC (CIG NSC), DCS (TSI NSS). The calling machine (receiver) would enter a T.30 holdup state after receiving the DIS signal from the answering machine (transmitter), and the user-supplied function would be called with *section* set to `T30H_SECTION_XMIT_DCS`.

The answering machine (transmitter) would enter a T.30 holdup state after receiving the DTC signal from the calling machine (receiver), and the user-supplied function would be called with *section* set to `T30H_SECTION_XMIT_DCS`. The calling machine (receiver) would enter a T.30 holdup state after receiving the DCS signal from the answering machine (transmitter), and the user-supplied function would be called with *section* set to `T30H_SECTION_RCV_DCS`.

When an application begins fax transmission, for example using *BfvFaxBeginSend*, the Bfv API normally sends the T.30 parameters to the channel at that time. When using the T.30 holdup feature, however, the Bfv API does not send the T.30 parameters at that time. When the Bfv API calls the user-supplied function, the application is expected to examine the received DIS, CSI, and NSF information and decide what to do. Accordingly, the application can decide to transmit (and what to transmit), to receive (if polling is involved), or to abort the call.

After deciding, the application will call the same functions it would call when starting a fax without T.30 holdup, including the original function used to start the fax procedure (in this case, *BfvFaxBeginSend*). The functions that the application can call at this time include *BfvFaxBeginSend*, *BfvFaxSetLocalId*, *BfvFaxBegin*, *BfvFaxBeginSendRaw*, *BfvFaxT30Params*, *BfvFaxSetNSF*, *BfvFaxBeginRaw*, *BfvFaxBeginSendTiff*, and *BfvFaxBeginTiff*. Whichever function the application calls will then send the T.30 parameters.

If the fax procedure began with one of the functions that permit polling (*BfvFaxBegin*, *BfvFaxBeginRaw*, or *BfvFaxBeginTiff*), the values passed when the function is called the second time determine whether the channel will initially transmit or receive.

The receiver's options when the Bfv API calls the user-supplied function are limited to examining the DCS, TSI, and NSS information, deciding whether or not to abort the call, and returning 0 or 1 as appropriate.

Under certain conditions, a T.30 holdup can also occur after a page has finished transmission. If a T.30 parameter (for example, resolution or width) changes between pages, the transmitter and receiver will each enter a holdup state. When this occurs, the transmitter should not call any of the starting fax procedure functions listed above. If the transmitter and receiver change directions (polling), they will each enter a holdup state in a way similar to the polling scenario described earlier. At that time, the new transmitter (formerly the receiver) must perform the same actions a transmitter would normally perform when it enters the holdup state.

The application can retrieve received DIS, DCS, and DTC information via the `LINE_DCS` and `LINE_DIS_DTC` macros and received CSI, TSI, CIG, NSF, and NSS information from the `INFO_RES` structure (filled by the last call to *BfvFaxGetRemoteInfo*).

The information that follows applies to `T30_HOLDUP_RCV_MCF` mode only.

After a fax page is transmitted, the transmitter sends one of several FSK signals (EOP, MPS, EOM, or their procedure interrupt or ECM equivalents) to the receiver. The receiver responds with a signal which is typically MCF, though it could also be RTN, RTP, PIN, or PIP.

If the transmitter sent MPS after it receives MCF, it normally begins transmitting data for a new page immediately.

In this scenario, the receiver will enter a T.30 holdup state after receiving the EOP, MPS, EOM, or equivalent, and the Bfv API will call the user-supplied function with *section* set to `T30H_SECTION_RCV_MCF`. The Bfv API delays calling the user-supplied function until the receive function in progress has returned all the data (*BfvFaxReceiveData*) or until it has called *fwrite* to write all the data (all other fax receive functions).

When the Bfv API calls the user-supplied function, the receiver's options are limited to deciding whether or not to abort the call, returning 0 or 1 as appropriate.

The Bfv API expects applications to use the `T30_HOLDUP_RCV_MCF` mode that permits the receiving application to delay sending the MCF signal until it has determined that the received fax is properly saved.

See Also

LINE_FAX_T30_RCV_MCF_FSK (lp)

Example

```
int holdup(lp, arg, section)
BTLINE *lp;
char *arg;
int section;
{
    printf("Channel %d, received holdup in %s section\n",
        LINE_UNIT_NUM(lp),
        (section == T30H_SECTION_XMIT_DCS)? "xmit" : "rcv");
    if (section == T30H_SECTION_XMIT_DCS)
    {
        struct args_fax args;
        BT_ZERO(args);
        args.s_ips = arg;
        BfvFaxBeginSend(lp, &args);
    }
    return(0);
}
main()
{
    struct args_fax args;

    ...
    BT_ZERO(args);
    args.holdup_mode = T30_HOLDUP_DCS;
    args.func = holdup;
    args.arg = (char *)ips;
    BfvFaxT30Holdup(lp, &args);
    BT_ZERO(args);
    args.s_ips = ips;
    BfvFaxBeginSend(lp, &args);

    ...
}
```

BfvFaxT30Params

Purpose

Sets the transmission bit rate, scan time, modulation type and line compression.

Syntax

```
void
BfvFaxT30Params              (lp, args)
   BTLINE                    *lp;
   struct args_fax_t30_params *args;
```

The structure contains the following fields.

Input Fields

```
unsigned bit_rate;
unsigned scan_time;
unsigned modulation_type;
unsigned line_compression;
unsigned iaf_bit_rate;
char xmt_level;
char rcv_level;
```

Output Fields

```
RES res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.bit_rate

Specifies the maximum transmission bit rate for non T.38 Internet Aware Fax (IAF) modulation types.

Valid values include:

BITRATE_2400	0x00
BITRATE_4800	0x02
BITRATE_7200	0x01
BITRATE_9600	0x03

BITRATE_12000	0x04
BITRATE_14400	0x05
BITRATE_16800	0x06
BITRATE_19200	0x07
BITRATE_21600	0x08
BITRATE_24000	0x09
BITRATE_26400	0x0A
BITRATE_28800	0x0B
BITRATE_31200	0x0C
BITRATE_33600	0x0D
BITRATE_GENERIC	0xFF

Note: TruFax® does not support values 0x06 through 0x0D.

args.scan_time

Specifies the minimum transmission scan time in milliseconds. Valid values include:

SCANTIME_0	7
SCANTIME_5	1
SCANTIME_10	2
SCANTIME_20	0
SCANTIME_40	4

args.modulation_type

Specifies a modulation type to use.
Valid values include:

MODULATION_GENERIC 0x00
: Any, no restriction.

MODULATION_V27 0x01
: V.27 only.

MODULATION_V29 0x02
: V.29 only.

MODULATION_V33 0x03
: V.33 only.

MODULATION_V17 0x04
: V.17 only.

`MODULATION_V34` 0x05

V.34 only. TruFax® does not support this value.

`MODULATION_IAF` 0x06

T.38 Internet Aware Fax (IAF).

`args.line_compression`

Allows restriction of the fax compression type used on the phone line. This value overrides the *line_compression* user configuration file parameter.

Valid values include:

`LINECOMPR_ANY` 0

No restriction.

`LINECOMPR_MH` 1

MH only.

`LINECOMPR_MR_MH` 2

MH or MR only.

`LINECOMPR_MMR_MR_MH` 3

MH or MR or MMR.

`args.iaf_bit_rate`

If *args.modulation_type* is set to MODULATION_IAF, this value specifies the maximum transmission bit rate in bits per second for T.38 Internet Aware Fax (IAF) modulation.

Range: 14400 to 2400000

`args.xmt_level`

Specifies a relative value which is additive to the *btcall.cfg* value xmt_level_override. This xmt_level value will either add or subtract from the transmit level. This value is in 0.5 dB units. If this value is positive it will add to the transmit level and if this value is negative it will subtract from the transmit level.

`args.rcv_level`

Specifies a relative value which is additive to the *btcall.cfg* value rcv_level_override. This rcv_level value will either add or subtract from the receive level. This value is in 0.5 dB units. If this value is positive it will add to the receive level and if this value is negative it will subtract from the receive level.

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures*, in this document.

Details

Call this function immediately before the function used to begin fax transmission or reception (for example, before *BfvFaxBeginSend* or *BfvFaxBeginRaw*).

Alternatively, when using infopkt-formatted data files, the application can include an `INFOPKT_T30_PARAMETERS` infopkt near the start of the infopkt stream to set these parameters.

When a call uses ECM, the scan time specification has no effect.

If ECM is disabled, then MMR fax compression on the line is unavailable.

Transmission or reception of a fax using T.38 Internet Aware Fax (IAF) modulation is only supported in the following configuration:

- SR140 only
- SR140 must have an IAF license
- ECM is enabled
- T.38 version must be 1 or higher

Example

```
struct args_fax_t30_params args;

BT_ZERO(args);
args.bit_rate = BITRATE_9600;
args.scan_time = SCANTIME_0;
args.modulation_type = MODULATION_GENERIC;
BfvFaxT30Params(lp, &args);
```

BfvFaxT4TimerParams

Purpose

Obtains the T4 duration, the T4 attempt, the current T4 timer value and the T4 timer expiration, or sets a new T4 timer value.

Syntax

```
void
BfvFaxT4TimerParams          (lp, args)
   BTLINE                    *lp;
   struct args_fax           *args;
```

The structure contains the following fields.

Input Fields

```
unsigned t4TimerValue;
```

Output Fields

```
unsigned t4TimerValue;
unsigned t4Duration;
unsigned t4Attempt;
unsigned t4Timeout;
RES *res;
```

Input

lp

Pointer to the BTLINE structure.

args

Pointer to an argument structure containing input and output fields.

args.t4TimerValue

If this input value is set to 0, then the function call will return the *t4TimerValue*, *t4Duration*, *t4Attempt* and *t4Timeout* from the event sent by the firmware. If this input value is set to a value other than 0, then the function call will set a new T4 timer value by sending a command to the firmware.

Output

Return value: None.

args.t4TimerValue

The current value of the T4 timer setting.

args.t4Duration

The duration to receive the last response or the T4 timeout value if no response is received.

args.t4Attempt

The attempt number which just took place for the command/response exchange.

args.t4Timeout

Indicates if a response was received or the T4 timer expired.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

The keyword *t4_timer_mode* needs to be set to a value of 2 in order for this function to be utilized. See the keyword *t4_timer_mode* in the Parameters for Technical Support Purposes section for more information. After a T4 adapt information event is sent by the firmware to the host, this function can be called after setting *args.t4TimerValue* to 0 to obtain the current T4 timer setting, the duration to receive the last response or the T4 timeout value if no response is received, the last attempt number, and the T4 timeout flag indicating if a response was received or the T4 timer expired. To set a new T4 timer setting, call this function after setting *args.t4TimerValue* to the desired value for the T4 timer setting. The T4 timer value should be set between 3,000 and 15,000 ms.

Example

```
BTLINE *lp;
void t4TimerProcessEvent(BTLINE *lp, struct args_packet
*args)
{
    struct args_fax args_fax;
    unsigned t4TimerValue;
    unsigned t4Duration;
    unsigned t4Attempt;
    unsigned t4Timeout;

    // Check to see if the event received is
    // from the fax facility and is the
    // T4 adaptation information event.
    if ((args->facility == MILL_FACILITY_FAX) &&
        (args->cmd_verb == MILL_VERB_EVENT) &&
       (args->cmd_specifier == FAX_T4_ADAPT_INFO_EVENT))
```

```
    {
        // Obtain the current T4 timer value,
        // the T4 duration to receive a response
        // or T4 timeout, the T4 attempt
        // and the T4 timeout indication.
        BT_ZERO(args_fax);
        BfvFaxT4TimerParams(lp, &args_fax);
        t4TimerValue = args_fax.t4TimerValue;
        t4Duration = args_fax.t4Duration;
        t4Attempt = args_fax.t4Attempt;
        t4Timeout = args_fax.t4Timeout;

       // Set the T4 timer value to a new value.
       // For example, the following checks to see
       // if the attempt just completed is 1 and
       // if so it will set the T4 timer value to
       // the existing T4 timer value plus 500 ms.
       // End user can change the following to set
       // the T4 timer value to a new value according
       // to some algorithm determined by the end user
        if (t4Attempt == 1)
        {
            BT_ZERO(args_fax);
            args_fax.t4TimerValue = t4TimerValue + 500;
            BfvFaxT4TimerParams(lp, &args_fax);
        }
    }
}
LINE_SET_INCOMING_CMD_FUNC(lp, t4TimerProcessEvent);
```

BfvFaxWaitForTraining

Purpose

Waits until either the `training_complete` or `direction_change` interrupt is received.

Syntax

```
void
BfvFaxWaitForTraining        (lp, args)
   BTLINE                    *lp;
   struct args_fax           *args;
```

The structure contains the following fields.

Output Fields

RES *res;*

Input

lp

Pointer to the `BTLINE` structure.

args

Pointer to an argument structure containing input and output fields.

Output

Return value: None.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

Call this function after *BfvFaxGetRemoteInfo*.

If the `training_complete` interrupt is received, the next step to send or receive a fax (for example, *BfvFaxSendPage* or *BfvFaxReceivePages*) must be initiated. If the `direction_change` interrupt is received (possible only if polling was enabled with the *BfvFaxBegin* or *BfvFaxBeginRaw* functions), the direction to send or receive a fax is reversed, and both *BfvFaxGetRemoteInfo* and *BfvFaxWaitForTraining* must be called again. Check the line state to determine the appropriate action.

This function is incorporated into the higher-level functions *BfvFaxSend*, *BfvFaxReceive*, and *BfvFaxPoll*.

When this function returns without error, the line state changes to either `FAX_MODE` for normal transmission or `TURNAROUND` for polling.

After completion of *BfvFaxWaitForTraining*, the application can access decoded values of the FSK message information exchanged between transmitter and receiver. See the `LINE_DCS` and `LINE_DIS_DTC` in *Macros on page 723* for more detailed information.

See Also

BfvFaxGetRemoteInfo

Example

```
BTLINE *lp;
struct args_fax args;

BT_ZERO(args);
BfvFaxWaitForTraining(lp, &args);
```

Macros

There are many important macros available to applications. For each of the following macros, *lp* is defined as **BTLINE** *.

LINE_DCS *(lp)*

Returns a pointer to a structure of type `dcs_info`, that contains decoded information from the DCS FSK message(s) sent from transmitter to receiver. The information is available after *BfvFaxWaitForTraining* returns. See the `dcs_info` structure definition for more information about the stored values. The definition appears in *dcs.h*. Applications must not modify the values in the structure since the Bfv API uses these values.

Since retraining may occur after the values are initially stored in the structure, the values in some of the fields may not reflect the actual transmission parameters. If this situation occurs, the values in the structure are updated at the time of retraining.

LINE_DIS_DTC *(lp)*

Returns a pointer to a structure of type `dis_dtc_info` that contains decoded information from the DIS or DTC FSK message(s) sent from receiver to transmitter. The information is available after *BfvFaxWaitForTraining* returns. See the `dis_dtc_info` structure definition for more information about the stored values. The definition appears in *dcs.h*. Applications must not modify the values in the structure since the Bfv API uses these values.

Since retraining may occur after the values are initially stored in the structure, the values in some of the fields may not reflect the actual transmission parameters. If this situation occurs, the values in the structure are updated at the time of retraining.

LINE_FAX_RES *(lp)*

Accesses a pointer to a preallocated buffer of type FAX_RES. This buffer stores PAGE_RES structures that are automatically allocated when a page is sent or received. The application may set the value of this macro to such a pointer to enable allocation and storage of PAGE_RES structures. The application must free these structures after use. After enabling this feature, set this macro to NULL to disable it.

Typically, this feature is enabled before beginning fax transmission or reception (for example, using *BfvFaxBeginSend*). If the high-level fax functions (for example, *BfvFaxSend*) are used, the FAX_RES structure supplied to that function takes precedence.

You should zero out (using memset) the FAX_RES structure between calls or before assigning it to this macro.

See *Volume 6, Appendix B, Bfv API Structures* for more information about these structure types. See also the *max_pagelist* parameter in *Volume 6, Appendix A, User-defined Configuration File.*

LINE_FAX_T30_RCV_MCF_FSK *(lp)*

This is intended for use with T.30 holdup RCV_MCF case. It returns the receive status FSK being sent to the transmitter. See *BfvFaxT30Holdup*.

LINE_FIRM_BITRATE *(lp)*

For compatibility only, this macro can be used to indicate the maximum bit rate the firmware and hardware support. The macro returns a fixed value depending on the features supported by the firmware/hardware. If fax is supported and V.34 is also supported, returns 33600; if fax only is supported, returns 14400; otherwise returns 0.

LINE_FONT_DOWNLOADED *(lp, font_no)*

Returns nonzero if a font was successfully downloaded to the module as the specified numbered font after firmware was downloaded.

Note: This macro returns a long (32-bit) value.

LINE_PAGE_CT *(lp)*

Accesses a page counter kept during fax transmission and reception. The value is initialized to 0 at the start of fax procedures and is incremented each time a page transmission or reception ends. The application can reset this counter.

Normally this macro is useful only when used in conjunction with the `LINE_SET_PAGE_COMPLETE_FUNC` or `LINE_SET_INCOMING_CMD_FUNC` macro (see *Determining Fax Status Information from an Application* in *Chapter 2* of the *Dialogic® Brooktrout® Fax Products SDK Developer Guide*, for more information).

LINE_SET_PAGE_COMPLETE_FUNC *(lp, func, arg)*

Sets up a user-supplied function that the Bfv API calls when it processes an end-of-page.

The Bfv API calls the user supplied function as:

```
void (*func)(BTLINE *lp, int stage, char *arg);
```

The *lp* argument contains the pointer to the line structure; the *stage* argument contains a value specifying the processing that has been performed by the Bfv API; the *arg* argument contains the supplied user-defined argument.

If the value of *stage* is 1, it indicates that the page completion has been processed but the confirmation value has not been received yet. If the value is 2, it indicates that the processing of the page completion and confirmation value is complete, and the `PAGE_RES` structure for the current page is ready.

A value of `NULL` for *func* disables this feature.

`LINE_PAGE_COMPLETE_FUNC` *(lp)*
`LINE_PAGE_COMPLETE_ARG` *(lp)*

Similar to `LINE_SET_PAGE_COMPLETE_FUNC`. Allows accessing of the page complete function and argument for setting or reading.

20 - TIFF-F Files Functions

This chapter provides descriptions and programming examples to help clarify the use of the functions used to process TIFF-F image files.

Functions used to manage fax transmission and reception for TIFF-F and other image formats are documented in *Fax Functions on page 630*.

Table 3 provides a list and brief description of all functions available to process TIFF_F image files.

TIFF-F Files Function Summary

The TIFF-F fax functions are used exclusively to send and receive TIFF-F files rather than formatted infopkt stream files. *Table 3* provides a brief summary of these functions. Detailed information about these functions begins on *page 783*.

Table 3. TIFF-F Function Summary

Function	Purpose	Page
BfvTiffClose	Closes an opened TIFF-F file.	*783*
BfvTiffOpen	Opens a TIFF-F file.	*785*
BfvTiffReadIFD	Reads the IFD of the current page in a TIFF-F file.	*787*
BfvTiffReadImage	Reads image data of current page, one buffer at a time.	*790*
BfvTiffReadRes	Interprets the IFD entry that contains the Y resolution tag; returns vertical resolution of current page.	*792*
BfvTiffWriteIFD	Writes the IFD of the current page in a TIFF-F file.	*794*
BfvTiffWriteImage	Writes image data of the current page to a TIFF-F file.	*797*
BfvTiffWriteRes	Writes the page resolution specifications data to a TIFF-F file; returns the offset of the location where this data is written.	*799*

BfvTiffClose

Purpose

Closes an opened TIFF-F file and frees all associated structure memory.

Syntax

```
int
BfvTiffClose                (args)
   struct args_tiff         *args;
```

The structure contains the following fields.

Input Fields

TFILE *tp;

Output Fields

RES *res;*

Input

args

Pointer to an argument structure containing input and output fields.

args.tp

*TFILE** pointer to the open file, returned by *BfvTiffOpen*.

Output

Return value:

0 The file was closed successfully.

non-0An error occurred in closing the file.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

The pointer may not be reused after a closing error value is returned.

See Also

BfvTiffOpen

Example

```
TFILE *tp;
struct args_tiff args;

BT_ZERO(args);
args.fname = "image.tif";
args.fmode = "r";
if ((tp = BfvTiffOpen(&args)) == NULL)
    fprintf(stderr,"Error opening TIFF file.\n");
else
{
    args.tp = tp;
    .
    .
    .
    BfvTiffClose(&args);
}
```

BfvTiffOpen

Purpose

Opens a TIFF-F file.

Syntax

```
TFILE *
BfvTiffOpen                    (args)
   struct args_tiff            *args;
```

The structure contains the following fields.

Input Fields

```
char *fname;
char *fmode;
```

Output Fields

```
RES res;
```

Modified Fields

tp

Input

args

Pointer to an argument structure containing input and output fields.

args.fname

Filename of the TIFF-F file.

args.fmode

Sets the mode to read or write.

"r" Read.

"w" Write.

Output

Return value:

*TFILE** pointer	Disk file was successfully opened.
NULL	An error occurred.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function retains the same resource sharing restrictions as the fopen call.

After opening a TIFF file for reading, the *BfvTiffReadIFD* function is normally called and then the *BfvTiffReadImage* function is called repeatedly. After opening a TIFF file for writing, normally the *BfvTiffWriteImage* function is called repeatedly and then the *BfvTiffWriteIFD* function is called repeatedly.

Both the current TIFF-F standard and the older TIFF Class F standards are supported for reading. When creating new files, only the current TIFF-F standard will be used. These files should be compatible with most earlier applications that used TIFF Class F.

See Also

BfvTiffClose

Example

```
TFILE *tp;
struct args_tiff args;

BT_ZERO(args);
args.fname = "image.tif";
args.fmode = "r";
if ((tp = BfvTiffOpen(&args)) == NULL)
    fprintf(stderr,"Error opening TIFF file.\n");
else
{
    args.tp = tp;
    .
    .
    .
    BfvTiffClose(&args);
}
```

BfvTiffReadIFD

Purpose

Reads the IFD of the current page in a TIFF-F file.

Syntax

```
int
BfvTiffReadIFD              (args)
   struct args_tiff         *args;
```

The structure contains the following fields.

Input Fields

```
TFILE *tp;
int (*func)(TFILE *tp, struct ifd_field *ifd_ptr,
   char *arg);
char *arg;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.tp

TFILE * pointer to a TIFF-F file.

args.func

A pointer to a user-supplied integer function that is called once for each IFD entry for the current page in the TIFF file. The value of *args.func* cannot be NULL.

If *args.func* returns 0, processing of the current IFD continues. If *args.func* returns a nonzero value, processing of the current IFD halts, and *BfvTiffReadIFD* indicates an error. For a description of how calls are made and of the arguments, see this function's *Purpose* section.

args.arg

A user-supplied argument for *args.func* which can be NULL.

Output

Return value:

0	No more pages are in the file.
>0	The IFD was successfully read.
<0	An error occurred.

args.res

A RES structure containing status information. The RES structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function is called directly after *BfvTiffOpen* and before calling *BfvTiffReadImage*. If the application elects to bypass reading the image after the IFD is read, *BfvTiffReadIFD* can be called again to proceed to the next page.

As it reads each IFD entry, this function calls the user-supplied function as:

```
(*args.func)(args.tp,ifd_ptr,args.arg)
```

The *ifd_ptr* argument is of type *struct ifd_field**, a pointer to a structure type containing the following. Some commonly used tags (symbols TAG_...), and field type definitions (symbols FT_. . .) are defined in *tiff.h*.

```
struct ifd_field {
short tag;                  /* Tag type, usually one of the
                            TAG_... values */
short field_type;           /* Field type, one of the FT_...
                            values */
unsigned count;             /* Number of values of the field
                            type */
MILL_PTR_INT_TYPE offset;   /* offset for values or the value(s)
                            themselves */
};
```

The argument *arg* is the user-supplied argument. After all IFD entries are read, *args.func* is called once with a NULL *ifd_ptr* value.

After receiving an *ifd_field* structure, the *offset* field contains either a value or an *offset* that indicates where the values are located. To read the value(s) at the offset, use fseek to move the current file position to the appropriate offset. Use the TIFF_FP (*tp*) file pointer if *tp* is the *TFILE* * pointer. Returning to the starting position after reading data at another offset is not necessary; the *BfvTiffReadIFD* function always reads from the correct position.

To read the resolution value, use the *BfvTiffReadRes* function.

If the *field_type* value is such that more than one value can fit into the *offset* field, the first value occupies the least significant portion of the *offset* field, the second value occupies the most significant portion of the *offset* field, and so on. This arrangement is followed whether the TIFF file byte-ordering format is Intel or Motorola.

The type of byte-ordering format becomes significant when the application reads data values at the offset specified by the *offset* field. Motorola format stores multibyte integers in the file in the opposite order from the internal representation of integers on Intel x86 and all Brooktrout-supported platforms. Therefore, the application will often need to reverse the byte order of the integer values read from the file before using them. The application can examine the *bytes_reversed* field of the *TFILE* structure to determine which byte-ordering format is used in the TIFF file; a nonzero value indicates Motorola format.

The user-supplied function must not call any function that causes a delay, such as waiting for a Touch-Tone for a nonzero timeout or going to sleep. All calls made within the user-supplied function must return immediately. Applications performing Touch-Tone detection must enable the detection before beginning the operation that uses the user-supplied function (for example, speech playback).

See Also

BfvTiffReadImage, *BfvTiffReadRes*

Example

See the *tiffdump.c* or *tstrip.c* applications in the app.src directory.

BfvTiffReadImage

Purpose

Reads the image data of the current page in a TIFF-F file, one buffer at a time.

Syntax

```
int
BfvTiffReadImage            (args)
   struct args_tiff         *args;
```

The structure contains the following fields.

Input Fields

```
TFILE *tp;
unsigned char *buf;
unsigned size;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.tp

TFILE * pointer to a TIFF-F file.

args.buf

Pointer to a user-allocated data buffer.

args.size

The size, in bytes, of the user-allocated buffer.

Output

ReturnValue:

0	Image data is completed.
>0	Number of bytes stored in *args.buf* on this call.
<0	An error occurred.

`args.res`

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function is called directly after *BfvTiffReadIFD*. Call this function repeatedly until the return value is 0.

TIFF-F pages may be internally constructed so that the image data is partitioned into strips. If a page that uses MMR data format is constructed in this way, there will be multiple MMR end-of-data markers (known as EOFBs), one after each strip of data.

In such cases, the data read for different strips cannot simply be concatenated together since an EOFB indicates an end of page. When transmitting such data via *BfvFaxSendData* the strips can be separated using *BfvFaxStripParams*.

See Also

BfvTiffReadIFD

Example

See the *tstrip.c* application in the *app.src* directory.

BfvTiffReadRes

Purpose

Interprets IFD entries from a TIFF-F file containing the tags for the X and Y resolution, and returns the resolution of the current fax page.

Syntax

```
int
BfvTiffReadRes              (args)
   struct args_tiff         *args;
```

The structure contains the following fields.

Input Fields

```
TFILE *tp;
struct ifd_field *horiz_ifd_field;
struct ifd_field *vert_ifd_field;
unsigned res_unit;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.tp

TFILE * pointer to a TIFF-F file.

args.horiz_ifd_field

Pointer to a filled-in *struct ifd_field* structure, read using *BfvTiffReadIFD*. The tag field of this structure must be TAG_XRESOLUTION.

args.vert_ifd_field

Pointer to a filled-in *struct ifd_field* structure, read using *BfvTiffReadIFD*. The tag field of this structure must be TAG_YRESOLUTION.

`args.res_unit`

The resolution unit value from the IFD entry with tag `TAG_RESOLUTIONUNIT`. This allows for TIFF files with resolutions expressed in units of either inches (as is standard) or centimeters. If left set to 0, inches are used.

Output

Return value:

≥0 Resolution value. See the *BfvFaxBeginSendRaw* function on *page 662* for resolution values.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function is provided as a convenience. It enables an application using *BfvTiffReadIFD* to easily compute the resolution of the current page. The application is freed from performing the low-level steps required for this computation.

The function interprets IFD entries from a TIFF-F file that contains the tag for the Y resolution (`TAG_YRESOLUTION`) and the tag for the X resolution (`TAG_XRESOLUTION`), and returns the resolution of the current fax page.

To interpret the IFD entry, *BfvTiffReadRes* uses the `fseek` function to move to the appropriate location in the TIFF file, reads two values, and examines their quotient. If the result is equal to or close to certain fixed values for normal or fine resolution, the appropriate value is returned.

See Also

BfvTiffReadIFD

Example

See the *tstrip.c* application in the *app.src* directory.

BfvTiffWriteIFD

Purpose

Writes the IFD of the current page to a TIFF-F file.

Syntax

```
int
BfvTiffWriteIFD              (args)
   struct args_tiff          *args;
```

The structure contains the following fields.

Input Fields

```
TFILE *tp;
struct ifd_field *ifd_field;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.tp

TFILE * pointer to a TIFF-F file.

args.ifd_field

Pointer to a structure of type *struct ifd_field*, which contains the `IFD` entry to write.

Output

Return value:

1 An error occurred.

0 IFD successfully written.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function is called repeatedly and directly after all calls to *BfvTiffWriteImage* until the entire IFD is written. After all IFD entries are written, this function must be called once with a `NULL` *ifd_field* value.

The *struct ifd_field* is shown below. Some commonly used tags (symbols `TAG_...`) and field type definitions (symbols `FT_...`) are defined in *tiff.h*.

```
struct ifd_field {
short tag;                    /* Tag type, usually one of the
                              TAG_... values */
short field_type;             /* Field type, one of the FT_...
                              values */
unsigned count;               /* Number of values of the field
                              type */
MILL_PTR_INT_TYPE offset;     /* offset for values or the value(s)
                              themselves */
};
```

When writing an *ifd_field* structure, the *offset* field contains either a value or an *offset* that indicates where the values are located. To write the value(s) elsewhere in the file, use `fseek` to move the current file position to the end of the file and write the values there. Note that the TIFF specification requires that all offset values be even. Use the `TIFF_FP` (*tp*) file pointer if *tp* is the *TFILE* * pointer. Returning to the starting position after writing data at another offset is not necessary; the *BfvTiffWriteIFD* function always writes from the correct position.

To write the resolution value, use the *BfvTiffWriteRes* function.

The IFD entries associated with tag types `TAG_STRIPOFFSETS` and `TAG_STRIPBYTECOUNTS` cannot be explicitly written; these entries are automatically put in the IFD based on the image previously written with the *BfvTiffWriteImage* function. For all other tag types, the user must decide what IFD entry/tag combinations to write.

BfvTiffWriteIFD automatically sorts the IFD entries in ascending tag order as required by TIFF specifications. Duplicate tag values are not permitted. The earliest IFD entry with a particular tag value is the one that is written. A maximum of forty different IFD entries can be written per `IFD`.

See Also

BfvTiffWriteImage, *BfvTiffWriteRes*

Example

```
TFILE *tp = BfvTiffOpen("image.tif", "w");
struct ifd_field ifd_entry;
struct args_tiff args;

/* Write the image */
...
/* Then write the IFD entries* /
/* This is just one of many entries that will be written */
ifd_entry.tag = TAG_COMPRESSION;
ifd_entry.field_type = FT_SHORT;
ifd_entry.count = 1;
ifd_entry.offset = 3;     /* Group 3 data */

BT_ZERO(args);
args.tp = tp;
args.ifd_entry = &ifd_entry;
BfvTiffWriteIFD(&args);

/* Write the rest of the entries */
...
args.ifd_entry = NULL;
BfvTiffWriteIFD(&args);
```

BfvTiffWriteImage

Purpose

Writes the image data of the current page to a TIFF-F file.

Syntax

```
int
BfvTiffWriteImage          (args)
   struct args_tiff        *args;
```

The structure contains the following fields.

Input Fields

```
TFILE *tp;
unsigned char *buf;
unsigned size;
```

Output Fields

```
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.tp

*TFILE** pointer to a TIFF Class F file.

args.buf

A buffer containing the image data to be written.

args.size

The number of bytes contained in *buf*.

Output

Return value:

0	Image successfully written.
>0	An error occurred.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function is called repeatedly and directly after *BfvTiffOpen* (before *BfvTiffWriteIFD* is called repeatedly).

The function must be repeatedly called with buffers of data until the entire image is written. After all image data is written, this function must be called once with a `NULL` *buf* value and a size of 0.

The original TIFF Class F standard required byte-aligned EOLs and no RTC. The more recent TIFF-F standard does not require these properties although they are permitted. If the data supplied to *BfvTiffWriteImage* does have byte-aligned EOLs, the RTC will automatically be removed.

See Also

BfvTiffWriteIFD

Example

```
FILE *fp = fopen("image.301", "w");
TFILE *tp = BfvTiffOpen("image.tif", "w");
int n;
unsigned char buf[1024];
struct args_tiff args;

BT_ZERO(args);
args.tp = tp;
args.buf = buf;

/* Write the image */

while((n = fread(buf,1,sizeof(buf),fp)) > 0)
{
    args.size = n;
    BfvTiffWriteImage(&args);
}
args.buf = NULL;
args.size = 0;
BfvTiffWriteImage(&args);

/* Then write the IFD entries */
...
```

BfvTiffWriteRes

Purpose

Writes the page resolution data for a page to a TIFF-F file and returns the offset of the location where this data is written.

Syntax

```
void
BfvTiffWriteRes              (args)
   struct args_tiff          *args;
```

The structure contains the following fields.

Input Fields

```
TFILE *tp;
int resolution;
```

Output Fields

```
long horiz_offset;
long vert_offset;
RES res;
```

Input

args

Pointer to an argument structure containing input and output fields.

args.tp

TFILE * pointer to a TIFF-F file.

args.resolution

The resolution of the page. See the *BfvFaxBeginSendRaw* function for resolution values.

Output

Return value: None.

`args.horiz_offset`

The offset in the TIFF file where the horizontal resolution data is located.

`args.vert_offset`

The offset in the TIFF file where the vertical resolution data is located.

args.res

A `RES` structure containing status information. The `RES` structure is documented in *Appendix B, Result Structures,* in this document.

Details

This function is provided as a convenience. It enables an application using *BfvTiffWriteRes* to easily write the resolution data. The application is freed from performing the low-level steps required to write the resolution data.

To include the vertical or horizontal resolution in the IFD for the current page, the application must pass *args.vert_offset* or *args.horiz_offset* to the *BfvTiffWriteIFD* function as the offset value of an IFD entry with tag `TAG_YRESOLUTION or TAG_XRESOLUTION`, respectively.

To write the page resolution data, *BfvTiffWriteRes* uses the *fseek* function to move to the end of the TIFF file and writes fixed values that correspond to the desired resolution.

See Also

BfvTiffWriteIFD

Macros

TIFF_FP *(tp)*

Returns the file pointer associated with the opened TIFF-F file; is suitable for use in functions such as *fseek*, *fread*, and *fwrite*.

Volume 5 - BSMI-Level Call Control and Call Switching

About this Volume

Volume 5, BSMI-Level Call Control and Call Switching, provides information about the following Boston Simple Message Interface (BSMI) Bfv API components:

- BSMI-Level Call Control functions
- Message structure
- R2 Signaling Protocol messages
- Local Exchange Carrier (LEC) Protocol messages
- Host to Module and Module to Host messages
- B-Channel and D-Channel Maintenance procedures

22 - BOSTON Simple Message Interface (BSMI)

This chapter describes the BSMI-level of call control functions an application uses to facilitate communication directly between the Dialogic® Brooktrout® module and the ISDN lines.

It has the following sections:

- *BSMI Installation*
- *BSMI-level Call Control functions listed alphabetically*
- *Error Return Values*
- *Firmware Download*
- *BSMI Use Examples*

The BOSTON Simple Message Interface (BSMI) consists of a comprehensive control mechanism for implementing advanced telecommunications services. The BSMI consists of two parts:

- A Bfv API library that allows the passing of messages between the applications and the firmware.
- The collection of the messages themselves.

This collection of messages is the interface to the ISDN component of the firmware and provides all the facilities for management, call control, and performance statistics monitoring. The ISDN

component of the firmware is common to the Dialogic® Brooktrout® platforms. Common software eases development, and the platforms share the exact type of interface through the BSMI.

The facilities provided within the low-level ISDN firmware of the Dialogic® Brooktrout® boards are consistent with capabilities originally introduced within the Instant ISDN Software (IISDN) of the Dialogic® Brooktrout® NetAccess series boards. Within this manual, any references to Instant ISDN Software are applicable to the firmware implemented on both series of Dialogic® Brooktrout® boards.

Control messages names in the BSMI are descriptive of the functions they serve and make it easier to develop applications.

The BSMI is based on the function call available in the Windows I/O Request Packet (IRP) driver library for the Dialogic® Brooktrout® T1/E1 WAN Controllers and BRI Network Adapters.

Having similar but differently named functions for boards allows an application to easily support simultaneous use of both series of Dialogic® Brooktrout® boards.

BSMI Installation

To install the BSMI, follow the installation instructions found in the *Dialogic® Brooktrout® Fax Products SDK Installation and Configuration Guide*. The following are the directories and files installed for BSMI:

	Directory	Contents
Windows	boston\bsmi.api\bapp.src\	getopt.c
	boston\bsmi.api\bapp.src\	getopt.h
	boston\bsmi.api\bapp.src\	vttyapp.c
	boston\bsmi.api\inc\	bsmilib.h
	boston\bsmi.api\inc\	iisdn.h
	boston\bsmi.api\inc\	r2_argentina.h
	boston\bsmi.api\inc\	r2_brazil.h
	boston\bsmi.api\inc\	r2_china.h
	boston\bsmi.api\inc\	r2_egypt.h
	boston\bsmi.api\inc\	r2_korea.h
	boston\bsmi.api\inc\	r2_mexico.h
	boston\bsmi.api\winnt\bapp.src\	makefile
Linux	boston/bsmi.api/bapp.src/	getopt.c
	boston/bsmi.api/bapp.src/	getopt.h
	boston/bsmi.api/bapp.src/	vttyapp.c
	boston/bsmi.api/inc/	bsmilib.h
	boston/bsmi.api/inc/	iisdn.h
	boston/bsmi.api/inc/	r2_argentina.h
	boston/bsmi.api/inc/	r2_brazil.h
	boston/bsmi.api/inc/	r2_china.h
	boston/bsmi.api/inc/	r2_egypt.h
	boston/bsmi.api/inc/	r2_korea.h
	boston/bsmi.api/inc/	r2_mexico.h
	boston/bsmi.api/linux/bapp.src/	makefile

BSMI Function Summary

Table 4 provides a brief summary of the BSMI functions.

Table 4. BSMI Function Summary

Function	Purpose	Page
BsmiClearVtty	Stops transmission/reception of data packets to/from firmware VTTY facility.	*808*
BsmiCloseAdapter	Closes and releases an open handle.	*809*
BsmiControlRead	Reads control message from the module.	*810*
BsmiControlWrite	Writes control message to the module.	*812*
BsmiLineAlert	Aborts a blocking ControlRead or ControlWrite function.	*813*
BsmiModuleList	Returns a list of hardware modules in the current system.	*815*
BsmiOpenAdapter	Returns a handle to the specific hardware module.	*817*
BsmiResetAdapter	Resets the ISDN component of the module associated with a handle.	*819*
BsmiSetVtty	Starts transmission/reception of data packets to/from firmware VTTY facility.	*821*
BsmiVttyRead	Reads data packet sent by firmware VTTY facility.	*822*
BsmiVttyWrite	Writes data packet to firmware VTTY facility.	*823*

BsmiClearVtty

Purpose

Stops transmission/reception of data packets to/from firmware VTTY facility.

Syntax

```
int
BsmiClearVtty                (handle)
   HANDLE                   handle;
```

The structure contains the following fields:

Input

handle

Handle to the control channel on a particular module.

Output

Return value:

0	Successful.
<0	Error, see *Error Return Values on page 824* for details.

Details

Terminates communication with the firmware VTTY facility.

See Also

BsmiSetVtty

Example

See vttyapp.c

BsmiCloseAdapter

Purpose

Closes and releases an open handle.

Syntax

```
int
BsmiCloseAdapter          (handle)
   HANDLE                 handle;
```

The structure contains the following fields:

Input

handle

Handle to the control channel on a particular module.

Output

Return value:

0	Successful.
<0	Error, see *Error Return Values on page 824* for details.

Details

Close and releases an open handle and associated *BTLINE* structures.

See Also

BsmiOpenAdapter

Example

```
HANDLE hp;

BsmiCloseAdapter (hp);
```

BsmiControlRead

Purpose

Reads a control message from the module.

Syntax

```
int
BsmiControlRead              (handle, buffer, args)
   HANDLE                    handle;
   L3_to_L4_struct           *buffer;
   struct args_bsmi          *args;
```

The structure contains the following fields:

Input Fields

```
unsigned timeout;
```

Input

handle

Handle to the control channel on a particular module.

buffer

Pointer to a buffer containing the *L3_to_L4_struct*. Refer to *Chapter , BSMI General Message Structure on page 844* for further details.

args

Pointer to a structure containing input and output fields.

args.timeout

Time to wait for incoming packet, in milliseconds.

Output

Return value:

0	Successful.
<0	Error, see *Error Return Values on page 824* for details.

Details

This function waits for any control message from the module.

You can specify timeout (optional). If *args* is `NULL`, the timeout is the maximum possible, `MILL_MAX_TIMEOUT`. If a timeout occurs, the function returns `ERROR_TIMEOUT`.

See Also

BsmiControlWrite

Example

```
HANDLE hp;
L3_to_L4_struct l34msg;
int i, err;
IISDNu8bit num_digits;
err = BsmiControlRead(hp, &l34msg, NULL);
if (!err)
{
    switch (l34msg.msgtype)
    {
      case L3L4mSETUP_IND:
        num_digits = l34msg->data.setup_data.called_party.num_digits;
        for (i = 0; i < num_digits; i++)
            printf ("%d\n", l34msg->data.setup_data.called_party.digits[i]);
        break;
    }
}
```

BsmiControlWrite

Purpose

Writes the control message to the module.

Syntax

```
int
BsmiControlWrite            (handle, buffer)
   HANDLE                   handle;
   L4_to_L3_struct          *buffer;
```

The structure contains the following fields:

Input

handle

Handle to the control channel on a particular module.

buffer

Pointer to a buffer containing the *L4_to_L3_struct*. Refer to *Chapter , BSMI General Message Structure on page 831* for further details.

Output

Return value:

0	Successful.
<0	Error, see *Error Return Values on page 824* for details.

Details

In abnormal circumstances, if the driver and/or firmware cannot receive the message, this function blocks until space is available. Use *BsmiLineAlert* to abort this function.

See Also

BsmiControlRead

Example

```
HANDLE hp;
L4_to_L3_struct l43msg;

BT_ZERO(l43msg);
l43msg.msgtype = L4L3mREQ_BOARD_ID;
BsmiControlWrite(hp, &l43msg);
```

BsmiLineAlert

Purpose

Aborts a blocking ControlRead or ControlWrite function.

Syntax

```
int
BsmiLineAlert                (handle, args)
   HANDLE                    handle;
   struct args_alert         *args;
```

The structure contains the following fields:

Input Fields

```
int type;
int alert_value;
```

Input

handle

Handle to the control channel on a particular module.

args

Pointer to a structure containing input and output fields.

args.type

Indicates the destination alert type. Possible values include:

BSMI_ALERT_READ

BSMI_ALERT_WRITE

args.alert_value

An alert value passed to the alerted channel. If left set to 0, a value of 1 is sent. See *BfvLineAlert* in *Volume 1, Chapter 2, Line Administration and Initialization*.

Output

Return value:

0	Successful.
<0	Error, see *Error Return Values on page 824* for details.

Details

This function calls *BfvLineAlert*. See *Volume 1, Chapter 2,* for specifics on *BfvLineAlert*.

This function causes a *BsmiControlWrite* or *BsmiControlRead* function to return immediately with a Return value of ERROR_OPERATION_ABORTED. The alert waits, for example, if a *BsmiLineAlert* is called with a type of BSMI_ALERT_WRITE, when no *BsmiControlWrite* is waiting. The next call to *BsmiControlWrite* immediately returns.

See Also

BsmiControlWrite, *BsmiControlRead*

Example

```
HANDLE hp;
args_alert args;

BT_ZERO(args);
args.type = BSMI_ALERT_READ;
BsmiLineAlert(hp,&args);
```

BsmiModuleList

Purpose

Returns a list of hardware modules in the current system.

Syntax

```
int
BsmiModuleList              (args)
   struct args_mod          *args;
```

The structure contains the following fields:

Input Fields

```
unsigned char *list;
int list_size;
```

Output Fields

```
int total;
```

Input

args

Pointer to a structure containing input and output fields.

args.list

Pointer to an array to hold hardware module identifiers. The list contains *args.total* hardware module identifiers and is zero terminated.

args.list_size

Size of the array used to store hardware module identifiers. Must include space for the zero termination.

Output

Return value:

0	Successful.
<0	Error, see *Error Return Values on page 824* for details.

args.total

Returns the total number of hardware modules.

Details

This function provides a list of hardware modules located in the current system.

If more hardware modules are detected than *list_size –1*, the function returns ERROR_TOO_MANY_MODULES.

See Also

BsmiOpenAdapter

Example

```
unsigned char list[30];
struct args_mod args;
int i;

BT_ZERO(args);
args.list = list;
args.list_size = sizeof(list);
BsmiModuleList(&args);
i = 0;
while (list[i] != 0)
{
   printf("Found module number: %x\n",list[i]);
   i++;
}
```

BsmiOpenAdapter

Purpose

Returns a handle to the specified hardware module.

Syntax

```
HANDLE
BsmiOpenAdapter              (bus_number, module, args)
   int                       bus_number;
   int                       module;
   struct args_open          *args;
```

The structure contains the following fields:

Input Fields

```
unsigned read_buf_size;
unsigned write_buf_size;
```

Input

bus_number

Must be BSMI_BUS_NUMBER.

module

Module number.

args.read_buf_size

Specifies a driver buffer size to be used when creating the application session for messages from the hardware module. This value is in effect only if greater than the default driver buffer size. Use this feature only in circumstances where extremely high traffic volume warrants.

args.write_buf_size

Specifies a driver buffer size used when creating the application session for messages going to the hardware module. This value is in effect only if greater than the default driver buffer size. Use this feature only in circumstances where extremely high traffic volume warrants.

Output

Return value:

On success: A pointer to the control channel for the module

On error: INVALID_HANDLE_VALUE

Details

Returns a pointer to the control channel on a particular module, initializing the appropriate *BTLINE* structures. The *bus_number* is defined for NetAccess Series compatibility purposes and must be set to `BSMI_BUS_NUMBER` (`0xFE`).

In Linux systems, *HANDLE* is defined as **`typedef void`** **HANDLE*.

For applications with a heavy call load (especially with large simultaneous bursts of calls), set both the *read_buf_size* and *write_buf_size* to a large value (for example: 200000 is generous).

See Also

BsmiCloseAdapter

Example

```
HANDLE hp;
int module;
struct args_open args;
module = 2;
hp = BsmiOpenAdapter(BSMI_BUS_NUMBER,module, &args);
```

BsmiResetAdapter

Purpose

This function resets the ISDN component of the module associated with *handle*.

Syntax

```
int
BsmiResetAdapter            (handle, args)
   HANDLE                   handle;
   struct args_reset        *args;
```

The structure contains the following fields:

Input Fields

```
unsigned char no_config;
char *config_file_name;
```

Output Fields

```
unsigned char facil_ver_major;
unsigned char facil_ver_minor;
unsigned char facil_ver_build;
```

Input

handle

Handle to the control channel on a particular module.

args.no_config

Disables telephony interface configuration when equal to 1.

args.config_file_name

Filename for the user configuration file. If set to NULL, the function uses the *btcall.cfg* file in the current directory. If the function does not find a *btcall.cfg* file, it skips configuration.

Output

Return value:

0	Successful.
<0	Error, see *Error Return Values on page 824* for details.

args.facil_ver_major

Major version of the ISDN facility on the module.

`args.facil_ver_minor`

Minor version of the ISDN facility on the module.

`args.facil_ver_build`

Build version of the ISDN facility on the module.

Details

Initializes the firmware ISDN facility, which clears all calls and active D-channel processing for all ports on the module.

The call verifies that the version of the Dialogic® Brooktrout® firmware ISDN facility is compatible with the host version of included *iisdn.h* header file. If incompatible, ERROR_REVISION_MISMATCH is returned.

If *args.no_config* is 0, the telephony configuration file specified in *args.config_file_name* is used to configure all telephony hardware units for the module specified by *handle* and set up any specified telephony connections. If *args.no_config* is 1, configuration is skipped.

Once connections indicated in the file are established, connections remain in effect until explicitly disconnected. Generally, connections specified in the file are intended as static connections for long term use.

Example

```
HANDLE hp;
struct args_reset args;

BT_ZERO(args);
BsmiResetAdapter(hp,&args);
printf("Module ISDN facility version is %d.%d.%d\n",
args.facil_ver_major, args.facil_ver_minor,
args.facil_ver_build);
```

BsmiSetVtty

Purpose

Starts transmission/reception of data packets to/from firmware VTTY facility.

Syntax

```
int
BsmiSetVtty                    (handle)
   HANDLE                      handle;
```

The structure contains the following fields:

Input

handle

Handle to the control channel on a particular module.

Output

Return value:

0	Successful.
<0	Error, see *Error Return Values on page 824* for details.

Details

Begins communication with the firmware VTTY facility.

See Also

BsmiClearVtty

Example

See vttyapp.c

BsmiVttyRead

Purpose

Reads data packet sent by firmware VTTY facility.

Syntax

```
int
BsmiVttyRead                    (handle, buffer, length)
   HANDLE                       handle;
   char                         *buffer;
   int                          *length;
```

The structure contains the following fields:

Input

handle

Handle to the control channel on a particular module.

buffer

Pointer to a buffer of size at least 1024 where incoming data will be stored.

length

Pointer to an int variable in which the amount of data stored in buffer will be stored.

Output

Return value:

0	Successful.
<0	Error, see *Error Return Values on page 824* for details.

Details

Waits for and reads data sent by the firmware VTTY facility. The timeout is the maximum possible, MILL_MAX_TIMEOUT. If a timeout occurs, the function returns ERROR_TIMEOUT.

See Also

BsmiVttyWrite

Example

See vttyapp.c

BsmiVttyWrite

Purpose

Writes data packet to firmware VTTY facility.

Syntax

```
int
BsmiVttyWrite                (handle, buffer, length)
   HANDLE                    handle;
   char                      *buffer;
   int                       *length;
```

The structure contains the following fields:

Input

handle

Handle to the control channel on a particular module.

buffer

Pointer to the buffer containing the data to write.

length

Pointer to an int variable containing the length of the data buffer buffer to write.

Output

Return value:

0	Successful.
<0	Error, see *Error Return Values on page 824* for details.

Details

Writes data to the firmware VTTY facility. In abnormal circumstances, if the driver and/or firmware cannot receive the message, this function blocks until space is available. Use *BsmiLineAlert* to abort this function.

See Also

BsmiVttyRead

Example

See vttyapp.c

Error Return Values

The BSMI uses low level function calls available in the Bfv API. If one of these function calls returns an error, the *RES* results structure is translated to a similar Winerror value, which is then negated and returned by the BSMI call.

In non-Windows operating systems, these return values are defined in *bsmi.h*. All operating systems return the value shown in the Value column of *Table 5*.

Table 5. BSMI Error Return Values

RES.status	RES.line_status	winerror.h definition	Value
BT_STATUS_OK		ERROR_SUCCESS	0L
BT_STATUS_ERROR	APIERR_UNCLASSIFIED	ERROR_EXTENDED_ERROR	1208L
BT_STATUS_ERROR	APIERR_FILEIO	ERROR_IO_DEVICE	1117L
BT_STATUS_ERROR	APIERR_FILEFORMAT	ERROR_FILE_CORRUPT	1392L
BT_STATUS_ERROR	APIERR_BOARDCAPABILITY	ERROR_NOT_SUPPORTED	50L
BT_STATUS_ERROR	APIERR_NOTCONNECTED	ERROR_NOT_CONNECTED	2250L
BT_STATUS_ERROR	APIERR_BADPARAMETER	ERROR_BAD_COMMAND	22L
BT_STATUS_ERROR	APIERR_MEMORY	ERROR_NOT_ENOUGH_ MEMORY	8L
BT_STATUS_ERROR	APIERR_BADSTATE	ERROR_CONNECTION_ UNAVAIL	1201L
BT_STATUS_ERROR	APIERR_TOOSOON	ERROR_CONNECTION_ REFUSED	1225L
BT_STATUS_ERROR	default	ERROR_CAN_NOT_COMPLETE	1003L
BT_STATUS_ERROR_ DIAL		ERROR_UNEXP_NET_ERR	59L
BT_STATUS_ERROR_ HANGUP		ERROR_GRACEFUL_ DISCONNECT	1226L
BT_STATUS_USER_ TERMINATED		ERROR_CANCELLATION	1223L
BT_STATUS_TIMEOUT		ERROR_TIMEOUT	1460L
BT_STATUS_ALERT		ERROR_OPERATION_ABORTED	995L

In addition, some functions might return the following:

winerror.h definition	Value
ERROR_REVISION_MISMATCH	1306L
ERROR_TOO_MANY_MODULES	214L
ERROR_INVALID_PARAMETERS	87L
ERROR_MOD_NOT_FOUND	126
ERROR_INVALID_MODULETYPE	190L

The *ResetAdapter* functions can also return an error value based from –0x20000000L. This value represents *args_admin.reset_status* returned from *BfvLineReset* logically ORed with –0x20000000L.

BSMI functions return –1, if the *handle* argument equals zero.

Firmware Download

Call one *BsmiOpenAdapter* function for each hardware module; it returns a handle used to reference all ports on that hardware module. A hardware module may contain no ports or multiple network interface ports. To determine the quantity and type of ports use either *BfvTelephGetInfo* or see the *L3L4mBOARD_ID* message response to the *L4L3mSEND_BOARD_ID* requests.

The structures transferred by *BsmiControlWrite* and *BsmiControlRead* (*L4_to_L3_struct* and *L3_to_L4_struct* respectively) are discussed in detail in *Chapter , BSMI General Message Structure on page 831*.

The individual network port on a particular hardware module can then be further referenced by the *lapdid* component of the common message header. There are two *lapdid*s for each network port, referencing the two HDLC channels available per port. The lowest numbered HDLC channel on each port is currently used for signaling purposes, for example:

lapd_id	0	2	4	6	...
T1/E1 Telephony Unit	1	2	3	4	...

BSMI Use Examples

Initialization and BSMI Message Sequence

This example shows an initialization and BSMI message sequence, matching that for the typical ISDN call scenario previously presented. It does not show an example of a real call.

- Returns a handle to the specified hardware module:

```
#define LAPDID_SPAN_A 0

HANDLE hp;
int module = 0x02;
unsigned short call_ref;
unsigned short l4_id = 1;
struct args_open args_open;
struct args_reset args_reset;
L4_to_L3_struct l43msg;
L3_to_L4_struct l34msg;

hp = BsmiOpenAdapter(BSMI_BUS_NUMBER,module, &args_open);
```

- Resets the ISDN component of the module associated with *handle*. The example obtains the hardware configuration from a valid configuration file in the current directory.

Note: The *teleph.cfg* is a deprecated configuration file. See *Volume 6, Appendix H* for more information:

```
BT_ZERO(args_reset);
BsmiResetAdapter(hp,&args_reset);
```

- Writes an enable-protocol message:

```
BT_ZERO(l43msg);
l43msg.lapdid = LAPDID_SPAN_A;
l43msg.msgtype = L4L3mENABLE_PROTOCOL;
l43msg.data.enable_protocol.level1.l1_mode = IISDNl1modHDLC;
l43msg.data.enable_protocol.level2.l2_mode = IISDNl2modLAP_D;
l43msg.data.enable_protocol.level2.dce_dte = IISDNdirUSER_SIDE;
l43msg.data.enable_protocol.level3.l3_mode = IISDNl3modQ931;
l43msg.data.enable_protocol.level3.q931_cnfg.switch_type = IISDNstUNKNOWN;
l43msg.data.enable_protocol.level3.q931_cnfg.variant = IISDNvarNET5;
l43msg.data.enable_protocol.level3.q931_cnfg.b_channel_service_state[0] =
   0xfffefffe; /* For E-1 configuration; For T-1 use 0x7fffff */
BsmiControlWrite(fd, & l43msg);
```

- Waits for Layer 2 to establish:

```
while (BsmiControlRead(hp, &l34msg, NULL)!= 0)
   if (l34msg.msgtype == L3L4mPROTOCOL_STATUS)
      if (l34msg. L3L4.pcol_stat_data.status == IISDNdsESTABLISHED)
      break;
```

- Writes a *L4L3mCALL_REQUEST* message:

```
BT_ZERO(l43msg);
l43msg.lapdid = LAPDID_SPAN_A;
l43msg.L4_ref = L4_id;
l43msg.msgtype = L4L3mCALL_REQUEST;
l43msg.data.call_req_data.call_type = IISDNcalltypVOICE;
l43msg.data.call_req_data.bchannel = 1;
l43msg.data.call_req_data.called_party.num_digits = 10;
l43msg.data.call_req_data.called_party.num_type = IISDNnumtUNKNOWN;
l43msg.data.call_req_data.called_party.num_plan = IISDNnumpUNKNOWN;
strncpy(&l43msg.data.call_req_data.called_party.digits[0]),
   "0123456789",10;
BsmiControlWrite(hp, &l43msg);
```

- Waits for a *L3L4mCONNECT* message:

```
while (BsmiControlRead(hp, &l34msg, NULL))
   if ((l34msg.L4_ref == L4_id) && (l34msg.msgtype == L3L4mCONNECT))
   {
      call_ref = l3l4msg.call_ref;
   break;
   }
```

- Writes a *L4L3mCLEAR_REQUEST* message to disconnect:

```
BT_ZERO(l43msg);
l43msg.msgtype = L4L3mCLEAR_REQUEST;
l43msg.lapdid = LAPDID_SPAN_A;
l43msg.L4_ref = L4_id;
l43msg.call_ref = call_ref;
BsmiControlWrite(hp, &l43msg);
```

- Waits for the acknowledgement:

```
while (BsmiControlRead(hp, &l34msg, NULL))
   if ((l34msg.L4_ref == L4_id) && (l34msg.msgtype ==
   L3L4mCLEAR_REQUEST))
      {
   call_ref = l3l4msg.call_ref;
   break;
      }
```

- Closes and releases the open handle:

```
BsmiCloseAdapter(hp);
```

BRI Protocol Stack Initialization

This example shows the initialization of the protocol stack on a BRI card. Once all the layers are established, call control is done in the same manner as PRI ISDN.

```
L4_to_L3_struct msg;

//set layer 1 for BRI
memset(&msg,0,sizeof(L4_to_L3_struct));
msg.msgtype = L4L3mSET_HARDWARE;
msg.lapdid = LAPDID_PORT_A;
msg.data.hardware_data.line_data[0].briL1_cmd = IISDNcmdL1_ACTIVATE;
msg.data.hardware_data.line_data[0].bri_l1mode = IISDN_l1mode_Terminal;
BsmiControlWrite(fd, &msg);

//set layer 2 and 3
memset(&msg,0,sizeof(L4_to_L3_struct));
msg.msgtype = L4L3mENABLE_PROTOCOL;
msg.lapdid = LAPDID_PORT_A;
msg.data.enable_protocol.level1.l1_mode = IISDNl1modHDLC;
msg.data.enable_protocol.level2.l2_mode = IISDNl2modLAP_D;
msg.data.enable_protocol.level2.par.lap.dce_dte = IISDNdirUSER_SIDE;
msg.data.enable_protocol.level2.par.lap.TEI_mode = 1;

//use auto-assigned TEI
// msg.lli = TO_IISDN_LLI(0,TEI);
//need only if TEI_mode = 0;
msg.data.enable_protocol.level3.l3_mode = IISDNl3modQ931;
msg.data.enable_protocol.level3.q931_cnfg.switch_type = IISDNstUNKNOWN;
msg.data.enable_protocol.level3.q931_cnfg.variant = IISDNvarNET3;
msg.data.enable_protocol.level3.q931_cnfg.basic_rate = 1;

// Put all channels on spans A into service
msg.data.enable_protocol.level3.q931_cnfg.b_channel_service_state[0] = 6;
msg.data.enable_protocol.level3.q931_cnfg.b_channel_service_state[1] = 6;
msg.data.enable_protocol.level3.q931_cnfg.suppress_auto_spid = 1;

//don't send SPID
BsmiControlWrite(fd, &msg);

//start layer 2
msg.msgtype = L4L3mENABLE_PROTOCOL;
msg.data.enable_protocol.command = IISDNepcmdDL_ESTABLISH;
BsmiControlWrite(fd, &msg);
```

23 - BSMI General Message Structure

This chapter describes the general structure of the BSMI control messages passed between the host and module.

It has the following sections:

- *BSMI Message Naming Convention*
- *BSMI Control Messages by Category*
- *L4L3 Message Common Header*
- *L3L4 Message Common Header*
- *L4 Reference and Call Reference*
- *Common Structures*

The host communicates with the Dialogic® Brooktrout® module through the Control Interface. The host application (referenced as L4) issues BSMI control messages to configure the module or to instruct it to perform a specific action, such as make a call, clear a call, or request the status of an interface.

The module (referenced as L3) issues BSMI control messages to inform the host of the status of the interface, call events, or an error condition.

Additional information on this interface is contained in the *iisdn.h* C header file. The *iisdn.h* file may change and some features might not match the current documentation. If you upgrade to newer versions, review the new *iisdn.h* files along with recompiling all drivers, applications, and utilities.

BSMI Message Naming Convention

BSMI control messages are identified using the following naming convention:

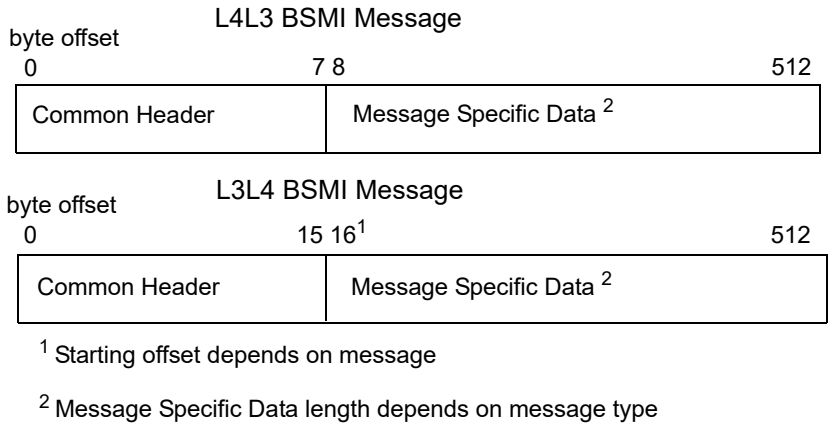
```
sender receiver m messageid
```

where:

`sender`	Identifies the source of the message. L4 refers to the host, L3 refers to the module.
`receiver`	Identifies the message destination. L4 refers to the host, L3 refers to the module.
`m`	Indicates that the message is a BSMI control message.
`messageid`	Identifies the specific message.

All BSMI control messages have the same general format, as illustrated in *Figure 1*. Messages from the host (L4) begin with a common data structure, or header. When the application orders an action, the header provides the information necessary to identify that message and the interface or call. See *L4L3 Message Common Header on page 842*.

A similar common header is used for all messages from the module (L3). Length of the message data varies depending on the specific message, but never exceeds 512 bytes. See *L3L4 Message Common Header on page 844*.



[1] Starting offset depends on message

[2] Message Specific Data length depends on message type

Figure 1. Control Message Structure

BSMI Control Messages by Category

There are several functional categories of BSMI control messages.

- *Management Messages*
- *Call Control Messages*
- *ISDN Supplemental and Miscellaneous Messages*

All the messages are arranged in alphabetical order by layer direction within protocol type:

- *Chapter , R2 Signaling Protocol with BSMI on page 876*
- *Chapter , LEC Protocols with BSMI on page 931*
- *Chapter , Host to Module (L4L3m) Messages on page 1005*
- *Chapter , Module to Host (L3L4m) Messages on page 1086*

Management Messages

Used for administration, configuration, and non-call related functions. For a summary of valid management messages, see:

- *Table 6, Management Messages (L4L3)*
- *Table 7 on page 835*

Table 6. Management Messages (L4L3)

Name	Meaning
L4L3mDISABLE_B_CHANNEL ID: 0xA3	Disables the specified B-channel or group of B-channels; places a B-channel into the out-of-service (near-end) state. For AT&T B-channel Maintenance only.
L4L3mDISABLE_CAS (R2) *L4L3mDISABLE_CAS (LEC)* ID: 0xC7	Disables the R2 protocol. This message is for any timeslot on an E1 span. Stops the LEC protocol. This message is issued for each logical channel available on the module. CAS protocols only (R2 and LEC).
L4L3mDISABLE_PROTOCOL ID: 0xA1	Disables Layers 1, 2, and 3 protocols and call processing for one or more 64K channels on a single module.

Table 6. Management Messages (L4L3) (Continued)

Name	Meaning
L4L3mENABLE_B_CHANNEL ID: 0xA2	Enables the specified B-channel; used to return a B-channel from out-of-service (near-end) to active state. For AT&T B Channel Maintenance only.
L4L3mENABLE_CAS (R2) *L4L3mENABLE_CAS (LEC)* ID: 0xC6	Starts the R2 protocol. This message is issued for each timeslot on the E1 span. Starts the LEC protocol. This message is issued for each logical channel available on the module. CAS protocols only (R2 and LEC).
L4L3mENABLE_PROTOCOL ID: 0xB6	Enables Layers 1, 2, and 3 protocols and call processing for one or more 64K channels on a single module.
L4L3mREQ_ABCD_DATA (R2) *L4L3mREQ_ABCD_DATA (LEC)* ID: 0xBC	Requests the module to return to the host all signaling bits from all B-channels in module. CAS protocols only (R2 and LEC).
L4L3mREQ_BOARD_ID ID: 0xC1	Requests module identification information from the module.
L4L3mREQ_CONFIGURATION ID: 0xEA	Retrieves parameters on a B-channel or line interface. CAS protocols (currently LEC only).
L4L3mREQ_PROTOCOL_STATUS ID: 0xA5	Reports current data link layer status for the specified D-channel.
L4L3mRESTART ID: 0xA4	Restarts the specified B-channel; idles and tears down any call on a B-channel.
L4L3mSET_CONFIGURATION ID: 0xE9	Sets parameters on a B-channel or line interface. CAS protocols (currently LEC only).

Table 7. Management Messages (L3L4)

Name	Meaning
L3L4mABCD_SIGNAL_DATA (LEC) ID: 0x35	Returns both the received and transmitted signaling bits for each B-channel on the module. CAS protocols only (R2 and LEC).
L3L4mACK_DOWNLOAD ID: 0x44	Indicates downloading of trunk-specific coefficients is successfully completed (currently applicable to analog modules only). CAS protocols (currently LEC only).
L3L4mACK_UPLOAD ID: 0x47	Returns both trunk-specific coefficients (currently only applicable to analog modules).
L3L4mB_CHANNEL_STATUS ID: 0x22	Indicates the status of the specified B-channel has changed.
L3L4mBOARD_ID ID: 0x3B	Sends the module identification to the host.
L3L4mCAS_SIGNALING_BIT_STATUS (R2) *L3L4mCAS_SIGNALING_BIT_STATUS (LEC)* ID: 0x4F	Notifies host that signaling bits for a B-channel have changed. CAS protocols only (R2 and LEC).
L3L4mCAS_STATUS (R2) *L3L4mCAS_STATUS (LEC)* ID: 0x40	Enable and disable CAS messages response. CAS protocols only (R2 and LEC).
L3L4mDISCONNECT ID: 0x21	Indicates a change in status on an enabled protocol stack (D-channels or B-channel).
L3L4mERROR ID: 0x20	Indicates either invalid information in the host specified L4L3 message or an error condition occurred.
L3L4mRESTART ID: 0x0D	Restarts the specified B-channel; idles and tears down any call on a B-channel.

Call Control Messages

Used for all ISDN call-related functions, such as setup, clearing, and event reporting. For a summary of valid call control messages, see:

- *Table 8, Call Control Messages (L4L3)*
- *Table 9 on page 838*

Table 8. Call Control Messages (L4L3)

Name	Meaning
L4L3mALERTING_REQUEST ID: 0x83	Informs the network an incoming call is ringing (send ALERTING message to the network).
L4L3mCALL_REQUEST ID: 0x81	Starts an outgoing call (send SETUP message to the network).
L4L3mCALL_PROCEEDING_REQUEST ID: 0x89	Notifies the switching equipment originating the call that the called party's line is free and being alerted (ringing). (R2) Send a CALL PROCEEDING message to the network; used with B-channel negotiation only.
L4L3mCAS_CHAN_BLOCK ID: 0xD6	Asserts blocking pattern on the line. Only valid from the idle state. Transitions the B-channel to the blocking state. CAS protocols (currently R2 only).
L4L3mCAS_CHAN_UNBLOCK ID: 0xD7	Asserts idle pattern on the line and return to the idle state. Only valid from the blocking state. CAS protocols (currently R2 only).
L4L3mCLEAR_REQUEST ID: 0x85	Clears (tears down) a call or refuses an incoming call (sends DISCONNECT, RELEASE, or RELEASE COMPLETE message to the network).
L4L3mCOLLECT_DIGITS (R2) *L4L3mCOLLECT_DIGITS (LEC)* ID: 0xC9	Acknowledges an incoming R2 MF digit and requests the next one. (R2) Instructs the module to start detecting digits. CAS protocols only (R2 and LEC).

Table 8. Call Control Messages (L4L3) (Continued)

Name	Meaning
L4L3mCONNECT_REQUEST ID: 0x84	Informs the network an incoming call has been answered (sends CONNECT message to the network).
L4L3mDIAL ID: 0xCA	Specifies digits dialed by the module, and the dialing method (DTMF, MF, or Analog Pulse). CAS protocols (currently LEC only).
L4L3mEND_DIAL ID: 0xBE	Notifies the module that the host has finished dialing digits. CAS protocols (currently LEC only).
L4L3mFACILITY_REQUEST ID: 0x86	Sends a FACILITY message to the network; used with Release Link Trunk signaling only.
L4L3mFEATURE_REQUEST ID: 0xB3	Requests ANI on Demand or Variabill feature for an incoming call; supported on AT&T #4ESS only.
L4L3mFORCE_CONNECTION_REQUEST ID: 0xEB	Notifies the module that the application has detected an answer for the current outbound call. Used when the protocol does not already provide that information. CAS protocols (currently LEC only).
L4L3mINFO_REQUEST ID: 0x8B	Dials a test R2 MF string of tones.
L4L3mPROGRESS_REQUEST ID: 0x82	Informs the network an incoming call is being processed (send PROGRESS message to the network).
L4L3mTX_HOOKFLASH ID: 0xC0	Transmits a hookflash if the line is currently offhook. Not all protocols support sending a hookflash. CAS protocols (currently LEC only).
L4L3mTX_WINK ID: 0xBD	Transmits a wink if the line is currently onhook. Not all protocols support sending a wink. CAS protocols (currently LEC only).

Table 8. Call Control Messages (L4L3) (Continued)

Name	Meaning
L4L3mUNIVERSAL ID: 0x88	Sends a custom or proprietary Q.931 message. Host application is responsible for constructing this message, and it has no effect on the call state.
L4L3mUSER_INFO ID: 0x87	Sends up to 130 bytes of information over the network without establishing a switched end-to-end connection.

Table 9. Call Control Messages (L3L4)

Name	Meaning
L3L4mALERTING (R2) *L3L4mALERTING (LEC)* *L3L4mALERTING (ISDN)* ID: 0x03	Indicates to the outbound side that the call is accepted by the remote end and the phone is ringing.
L3L4mANI ID: 0x31	Provides the result of the ANI on Demand request; generated in response to an *L4L3mFEATURE_REQUEST* message.
L3L4mBILLING_STATUS ID: 0x30	Provides the result of the billing change request (Variabill), generated in response to an *L4L3mFEATURE_REQUEST* message.
L3L4mCALL_PROC_SENT ID: 0x38	Reports the transmission of the call proceeding.
L3L4mCALL_PROCEEDING ID: 0x3A	Indicates that a call proceeding message was received.
L3L4mCALLER_ID_DETECTED ID: 0x59	Notifies host that called ID information is detected on the line.
L3L4mCAS_CHAN_BLOCKED ID: 0x4B	Indicates either a response to a block message or an asynchronous event that the line has been blocked by the remote end. CAS protocols (currently R2 only).

Table 9. Call Control Messages (L3L4) (Continued)

Name	Meaning
L3L4mCAS_CHAN_UNBLOCKED ID: 0x4C	Indicates either a response to an unblock message or an asynchronous event that the line was unblocked by the remote end. CAS protocols (currently R2 only).
L3L4mCLEAR_REQUEST (R2) *L3L4mCLEAR_REQUEST (LEC)* *L3L4mCLEAR_REQUEST (ISDN)* ID: 0x06	Indicates response to a CLEAR_REQUEST message. This event is not received until the remote end is idle. (R2) Notifies the host that the line is idle and ready for another call (inbound or outbound). (LEC)
L3L4mCLEAR_WITH_RESTART_REQUEST ID: 0x08	Indicates a call has been cleared due to a network detected error. The B-channel involved in the call has been restarted.
L3L4mCONN_ACK_IND ID: 0x0C	Received connection acknowledgement.
L3L4mCONNECT (R2) *L3L4mCONNECT (LEC)* *L3L4mCONNECT (ISDN)* ID: 0x04	Notifies the host that the ongoing outbound call is answered.
L3L4mDISCONNECT (R2) *L3L4mDISCONNECT (LEC)* *L3L4mDISCONNECT (ISDN)* ID: 0x05	Indicates that the remote end has disconnected. Notifies the host that the network released the line, either initiating the call disconnection procedure or in response to a disconnection initiated by the module.
L3L4mPRE_SEIZE (R2) *L3L4mPRE_SEIZE (LEC)* ID: 0x33	Indicates that the line was seized and that an incoming call is in progress. (R2) Notify host that the line is seized for an incoming call, and the handshaking necessary to proceed with the call is taking place. (LEC) CAS protocols only (R2 and LEC).

Table 9. Call Control Messages (L3L4) (Continued)

Name	Meaning
L3L4mPROGRESS (LEC) *L3L4mPROGRESS (ISDN)* ID: 0x02	Notifies the application that the procedure for establishing an outbound call (initiated by an L4L3mCALL_REQUEST message) is in progress.
L3L4mRAW_QDATA ID: 0x16	Contains up to 256 bytes of undecoded Q.931 packets; this message is usually generated in tandem with an ISDN L3L4m message and follows the L3L4 message, if any.
L3L4mSETUP_IND (R2) *L3L4mSETUP_IND (LEC)* *L3L4mSETUP_IND (ISDN)* ID: 0x01	Notifies the host of an incoming call. The called and calling party numbers are contained within the L3L4 data structure.
L3L4mSTATUS_IND ID: 0x07	Provides status information for a call, usually indicating a remote call state mismatch.
L3L4mUNIVERSAL ID: 0x17	Sends a custom or proprietary Q.931 message to the host. It has no effect on the call state.
L3L4mUSER_INFO ID: 0x09	Indicates receipt of a USER INFO message, containing up to 130 bytes of information.

ISDN Supplemental and Miscellaneous Messages

ISDN uses supplemental and miscellaneous messages for activities outside standard ISDN call-related functions, such as ITU-T suspend/resume functions and the passing of undecoded Q.931 packets from the network to the host. See *Table 10* for more information.

Table 10. ISDN Supplemental and Miscellaneous Messages (L3L4)

Name	Meaning
L3L4mANI ID: 0x31	Provides the result of the ANI on Demand request; generated in response to an *L4L3mFEATURE_REQUEST* message.
L3L4mBILLING_STATUS ID: 0x30	Provides the result of the billing change request (Variabill), generated in response to an *L4L3mFEATURE_REQUEST* message.
L3L4mRAW_QDATA ID: 0x16	Contains up to 256 bytes of undecoded Q.931 packets; this message is usually generated in tandem with an ISDN L3L4m message and follows the L3L4 message, if any.
L3L4mUNIVERSAL ID: 0x17	Sends a custom or proprietary Q.931 message to the host. It has no effect on the call state.

L4L3 Message Common Header

Description

The L4L3 message common header is used for all control messages passed from the host to the Dialogic® Brooktrout® module. It is eight bytes in length and has the structure described below.

All eight bytes of the common header must be present in each L4L3 message.

L4L3 messages are described in:

- *Chapter , R2 Signaling Protocol with BSMI on page 876*
- *Chapter , LEC Protocols with BSMI on page 931*
- *Chapter , Host to Module (L4L3m) Messages on page 1005*

Input Fields

```
unsigned char lapdid;
unsigned char msgtype;
unsigned short L4-ref;
unsigned short call_ref;
unsigned short lli;
```

Input

lapdid

LAP-D ID. A number from 0 to 63 that identifies the physical HDLC channel to be used.

msgtype

Message ID. Identifies the message being sent. Refer to *Table 6 on page 833* for valid L4L3 message IDs.

L4-ref

L4 Reference. Reference value assigned to outgoing calls by the host application for tracking purposes. The host defines the numbering scheme, but the value must be unique for each active call. The module references this number in all L3L4 response messages.

See *L4 Reference Value on page 846* and *Relationship between L4 Reference and Call Reference on page 848* for more information.

call_ref

Call Reference. Reference value assigned on a per call basis by the module. This value identifies the call for which the host is issuing an ISDN Call Control message. Use a value of 0x0000 for L4L3 Management messages and an initial *L4L3mCALL_REQUEST*.

See *Call Reference Value on page 847* and *Relationship between L4 Reference and Call Reference on page 848* for more information.

lli

Logical Link ID or DLCI. Used for LAP-D data connections. For other connection types, set these bytes to 0x0000. The format for the Logical Link ID is shown in *Figure 2 on page 849*.

See *Logical Link ID or DLCI on page 849* for more information.

L3L4 Message Common Header

Description

The L3L4 message common header is used for all control messages passed from the Dialogic® Brooktrout® module to the host. It is 16 bytes in length and has the structure described below.

All 16 bytes of the common header are present in each L3L4 message.

L3L4 messages are described in:

- *Chapter , R2 Signaling Protocol with BSMI on page 912*
- *Chapter , LEC Protocols with BSMI on page 931*
- *Chapter , Module to Host (L3L4m) Messages on page 1086*

Input Fields

```
unsigned char lapdid;
unsigned char msgtype;
unsigned short L4-ref;
unsigned short call_ref;
unsigned char bchannel;
unsigned char interface;
unsigned short lli;
```

Input

lapdid

LAP-D ID. A number from 0 to 63 that identifies the physical HDLC channel to be used.

msgtype

Message ID. Identifies the message being sent. Refer to *Table 6 on page 833* for valid L4L3 message IDs.

L4-ref

L4 Reference. Reference value assigned to outgoing calls by the host application for tracking purposes. The host defines the numbering scheme, but the value must be unique for each active call. The module references this number in all L3L4 response messages.

See *L4 Reference Value on page 846* and *Relationship between L4 Reference and Call Reference on page 848* for more information.

call_ref

Call Reference. Reference value assigned on a per call basis by the module. This value identifies the call for which the host is issuing an ISDN Call Control message. Use a value of 0x0000 for L4L3 Management messages and an initial *L4L3mCALL_REQUEST*. Refer to *page 847* for more information about *call_ref* values.

See *Call Reference Value on page 847* and *Relationship between L4 Reference and Call Reference on page 848* for details.

bchannel

B-channel. Identifies the channel. Channels are numbered as listed below:

ISDN 23B+D	Channels numbered 1 - 23. Channel 24 reserved for signaling.
ISDN 30B+D	Channels numbered 1 - 15, 17 - 31 (Channel 16 reserved for signaling). Channel 0 reserved for telemetry.
ISDN NFAS	Channels numbered 1 - 24; the interface field value indicates the span where the channel resides,

interface

Non-Facility Associated Signaling (NFAS) interface. Use a default value of 0xFF if NFAS is not configured. Interfaces are numbered from 0 – 19.

lli

Logical Link ID or DLCI. Used for LAP-D data connections. For other connection types, set these bytes to 0x0000. The format for the Logical Link ID is shown in *Figure 2 on page 849*.

See *Logical Link ID or DLCI on page 849* for more information.

L4 Reference and Call Reference

The BSMI control interface uses two types of reference values in call handling: L4 Reference and Call Reference. Together, they provide both the host and the Dialogic® Brooktrout® module with a method of indicating the specific call for a Call Control message. *L4 ref* and *call_ref* are included in the common header for both L4L3 and L3L4 messages.

Refer to *L4L3 Message Common Header on page 842* and *L3L4 Message Common Header on page 844* for more information.

L4 Reference Value

The L4 Reference value is a unique number assigned to a call by the host application and exists for the duration of the call. Dialogic® Brooktrout® firmware stores the *L4_ref* upon receipt of any and every call control message. The Dialogic® Brooktrout® module uses this value in the following manner:

- *Outgoing calls* – The host must provide an *L4_ref* when it initiates a call using the *L4L3mCALL_REQUEST* message. All subsequent L4L3 Call Control messages for this call must use *L4_ref*. The Dialogic® Brooktrout® module stores *L4_ref* for use in all L3L4 messages pertaining to that call. *L4_ref* is especially important for outgoing B-channel negotiation since it is the only way to reference the call before the B-channel is assigned.
- *Incoming calls* – After receiving an *L3L4mSETUP_IND* (incoming call), the host is expected to provide an L4 reference value in its first L4L3 response to the card. In this initial message, the Dialogic® Brooktrout® module uses a value of 0xFFFF for *L4_ref*. If no *L4_ref* is provided, this module assigns a value of 0xFFFF for use in subsequent messages.

An *L4_ref* is not required for Management messages (*L4_ref* set to 0xFFFF as a default).

Call Reference Value

The call reference value is a unique number assigned to a call by the Dialogic® Brooktrout® module. The module uses this value in the following manner:

- *Outgoing calls* – The host must use a *call_ref* of 0x00 in the initial *L4L3mCALL_REQUEST* message since this number is assigned by the module. If no errors occur, the module returns *call_ref* to the host in the initial PROGRESS, ALERTING, or CONNECT message. All subsequent L4L3 Call Control messages from the host must include *call_ref*. L4L3 Call Control messages received without the proper call reference value cause an *L3L4mERROR* message with an error code of L3L4errCALL_REF_ERROR (06).
- *Incoming calls* – The module assigns a call reference value to each incoming call request from the network. The application passes this value to the host in an *L3L4mSETUP_IND* message. All subsequent L4L3 ISDN Call Control messages from the host must include this *call_ref*. L4L3 ISDN Call Control messages received without the proper call reference value cause an *L3L4mERROR* message with an error code of L3L4errCALL_REF_ERROR (06).

A call reference value is not required for Management messages (*call_ref* set to 0 as a default).

Relationship between L4 Reference and Call Reference

Typically, Dialogic® Brooktrout® firmware tracks calls using *call_ref*. When the host issues an L4L3m call control message, Dialogic® Brooktrout® firmware searches its call record data structure for the *call_ref* matching the *call_ref* received in the common header. If no matching *call_ref* is found, an *L3L4mERROR* message is generated with an error code of L3L4errCALL_REF_ERROR (06) indicating no record of the call can be found.

The *L4_ref* value is used to identify a call in the following cases:

- The application determines an outgoing call is to be abandoned prior to receiving a response from the module. In this case, *L4_ref* used in the initial *L4L3mCALL_REQUEST* message is used to identify the call; *call_ref* is zero. Dialogic® Brooktrout® firmware searches the call record data structures for a value matching the *L4_ref* received from the host. If no matching *L4_ref* is found, an *L3L4mERROR* message is generated with an error code of L3L4errCALL_REF_ERROR (06) indicating no record of the call can be found.
- When using B-channel negotiation for outgoing calls, the *L4_ref* value is the only way the call can be identified by the module in the first response to an *L4L3mCALL_REQUEST*. Subsequent L3L4m messages contain the appropriate *call_ref* value.

Logical Link ID or DLCI

The Logical Link ID is equivalent to the ISDN LAP-D/Q.921 Data Link Connection Identifier (DLCI). The DLCI consists of two bytes that uniquely identify a logical connection. The two main components of the DLCI are:

- *Terminal Endpoint Identifier (TEI)*

  The high level bits of the least significant byte (LSB), identifying the connected terminal equipment.

- *Service Access Point Identifier (SAPI)*

  The high six bits of the most significant byte (MSB), identifying the Layer 3 user of LAP-D.

The DLCI has the format shown in *Figure 2*.

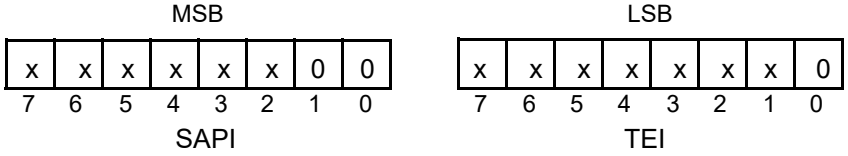


Figure 2. DLCI Format

The following excerpts from *iisdn.h* illustrate sample macros that can be used by the host for constructing the LLI/DLCI.

```
* For Q.921, the user might use the following macro to specify LLI in
* an ENABLE_PROTOCOL message:
* #define TO_IISDN_LLI(SAPI,TEI) (((SAPI) << 10) | ((TEI) << 1))
```

Table 11 lists the LLI values to be used for the various protocol configurations. For more information on protocol configuration, refer to *L4L3mENABLE_PROTOCOL on page 1033*.

Table 11. Protocol Mode and Corresponding LLI Values

Protocol Mode	Recommended LLI Value	
	iisdn.h define	Hex Value
LAP-D	use macro shown above	
Q.931 PRI and BRI point-to-multipoint	IISDN_LLI_CNTL	0x0000
Q.931 BRI point-to-point	use macro shown above	

Common Structures

ISDN Supplemental and Miscellaneous messages make use of common structures to initiate or report call status. This section defines these common structures referred to in the message structure definitions found in the following chapters:

- *Chapter , Host to Module (L4L3m) Messages on page 1005*
- *Chapter , Module to Host (L3L4m) Messages on page 1086*

The common structures include:

- *Alerting and Connecting Data Message (IISDN_AL_CON_DATA) on page 851*
- *Call ID (IISDN_CALL_ID) on page 854*
- *Called Party (IISDN_CALLED_PARTY) on page 855*
- *Calling Party (IISDN_CALLING_PARTY) on page 857*
- *Cause Data (IISDN_CAUSE) on page 860*
- *Connected Address (IISDN_CONNECTED_ADDRESS) on page 863*
- *Information Element (IISDN_IE_STRUCT) on page 865*
- *Progress Indication (IISDN_PROGRESS) on page 867*
- *Q.933 DLCI Negotiation (IISDN_Q922_DLCI) on page 870*
- *Redirecting Number (IISDN_REDIRECT_NUM) on page 871*
- *User Info (IISDN_USER_INFO) on page 875*

Dialogic® Brooktrout® firmware uses the data that populates these structures to provide call routing and billing information to the network in the format appropriate for the service supporting the Dialogic® Brooktrout® module. This data includes ISDN Information Elements (IEs) required for Q.931 connections.

Alerting and Connecting Data Message (IISDN_AL_CON_DATA)

Description

This message alerts and connects messages to and from the network. The Alerting and Connect Data Message structure detailed below can be found in the following messages:

- *L4L3mALERTING_REQUEST*
- *L4L3mCONNECT_REQUEST*
- *L3L4mALERTING*
- *L3L4mCONNECT*

Input Fields

```
IISDN_REDIRECT_NUM redirect_num;
unsigned short x25_tx_pktsize;
unsigned short x25_rx_pktsize;
unsigned char bchannel;
unsigned char iface;
IISDN_USER_INFO user_info;
IISDN_Q922_DLCI fr_dlci;
IISDN_CONNECTED_ADDRESS connected_address;
IISDN_PROGRESS_IND progress_ind;
IISDN_CALL_ID call_id;
IISDN_IE_STRUCT ie;
```

Input

redirect_num

Redirecting Number Structure. See *Redirecting Number (IISDN_REDIRECT_NUM) on page 871*.

x25_tx_pktsize

X.25 Transmit Packet Size. Unused in this message; set this field to 0x00

x25_rx_pktsize

X.25 Receive Packet Size. Unused in this message; set this field to 0x00

bchannel

B-channel. Identifies the channel to be used for the call. In this message use *bchannel* only when B-channel negotiation is enabled by setting *b_chan_negot* in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message. If the feature is not enabled or a *L4L3mCALL_PROCEEDING_REQUEST* message is already issued for this call, *bchannel* is ignored and set to 0x00. Channels are numbered as listed below:

ISDN 23B+D	Channels numbered 1 - 23.
ISDN 30B+D	Channels numbered 1 - 15, 17-31 (channel 16 reserved for signaling).
ISDN NFAS	Channels numbered 1 - 24; the interface field value indicates the span where the channel resides.
ISDN H0	Interfaces numbered either 1 - 3 (23B+D) or 1 - 4 (NFAS); supported for 384K ISDN service on AT&T 4ESS only.

iface

Non-Facilities Associated Signaling (NFAS) Interface. Indicates the channel's span specified in *bchannel*. The host application must maintain the mapping of span (line) to interface. In this message use *iface* only when both B-channel negotiation (*b_chan_negot)* and NFAS (*nfas)* is enabled in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message. If neither field is enabled, *iface* is ignored and set to 0x00. If B-channel negotiation is enabled and NFAS is not in use, set *iface* to 0xFF.

user_info

See *User Info (IISDN_USER_INFO) on page 875*.

fr_dlci

See *Q.933 DLCI Negotiation (IISDN_Q922_DLCI) on page 870*.

connected_address

See *Cause Data (IISDN_CAUSE) on page 860*.

progress_ind

See *Progress Indication (IISDN_PROGRESS) on page 867*.

call_id

See *Call ID (IISDN_CALL_ID) on page 854*.

ie

See *Information Element (IISDN_IE_STRUCT) on page 865*.

Call ID (IISDN_CALL_ID)

Description

Call ID provides information necessary to support Release Link Trunk (RLT) signaling on PRI spans connected to DMS-100 and DMS-250 switches. Since this method of call deflection is used to forward an incoming call to another destination without allocating a B-channel, the Call ID is the only way to identify this call.

The Call ID structure detailed below is used in the following messages:

- *L4L3mALERTING_REQUEST*
- *L4L3mFACILITY_REQUEST*
- *L4L3mPROGRESS_REQUEST*
- *L3L4mALERTING*
- *L3L4mCONNECT*
- *L3L4mPROGRESS*

Input Fields

```
unsigned char length;
unsigned char call_id [IISDN_MAX_CALL_ID];
```

Input

length

Length of Call ID. Specifies the number of ASCII (IA5) digits contained in the *call_id* array.

call_id

Call ID. An array of up to 31 ASCII (IA5) digits used to identify this call. When present in an L3L4 message, this value indicates the call ID number provided by the network in a FACILITY IE.

Called Party (IISDN_CALLED_PARTY)

Description

Called Party provides information necessary to route the call and/or construct the Called Party Number IE. The application uses this information to indicate the destination of the call and contains the following components:

- Number of digits in the destination telephone number
- Type of number (usually National for North American numbers)
- Numbering plan used (typically ISDN)
- Destination telephone number

The Called Party structure detailed below is used in the following messages:

- *L4L3mCALL_REQUEST*
- *L4L3mINFO_REQUEST*
- *L3L4mSETUP_IND*

Input Fields

```
unsigned short num_digits;
unsigned char num_type;
unsigned char num_plan;
unsigned char digits [IISDN_MAX_DIGITS];
```

Input

num_digits

Number of Digits. Specifies the number of digits in the Called Party Number. The application determines valid Calling Party numbers by the current network dialing plan. This number must be less than 0x18.

num_type

Number Type. Corresponds to the Called Party Number Types as defined by ITU-T. This field is usually specified as IISDNnumtNATIONAL (0x02). Possible values include:

IISDNnumtUNKNOWN 0x00

Unknown number type, used when there is no knowledge of the type of number.

IISDNnumtINTERNATIONAL 0x01

International number.

IISDNnumtNATIONAL 0x02

National number.

IISDNnumtSUBSCRIBER 0x04

Subscriber number.

IISDNnumtABBREVIATED 0x06

Abbreviated number.

num_plan

Numbering Plan Identification. Corresponds to the Called Party numbering plan identification as defined by AT&T. This field is usually specified as IISDNnumpISDN (0x01). Possible values include:

IISDNnumpUNKNOWN 0x00

Unknown Numbering Plan.

IISDNnumpISDN 0x01

Numbering per ISDN Recommendation E.164/E.163.

IISDNnumpTELEPHONY 0x02

Standard telephone numbering plan.

IISDNnumpPRIVATE 0x05

Private numbering plan.

digits

Digits. Called Party number in ASCII digits. Number of digits in this string must equal the number of digits specified in *num_digits*. Eight-bit values in this field must be between 0x30 and 0x39 (ASCII 0 through 9).

Calling Party (IISDN_CALLING_PARTY)

Description

Calling Party provides information necessary to identify the originator of the call and/or construct the Calling Party Number IE. The application uses this information to indicate the origin of the call (originating number) and contains the following components:

- Number of digits in the originating telephone number (or zero if the number is not available)
- Type of number (usually National for North American numbers)
- Numbering plan used (typically ISDN)
- Optional Presentation indicator
- Optional Screening indicator
- Originating telephone number, if available

The Calling Party structure detailed below is used in the following messages:

- *L4L3mCALL_REQUEST*
- *L3L4mANI*
- *L3L4mSETUP_IND*

Input Fields

```
unsigned short num_digits;
unsigned char num_type;
unsigned char num_plan;
unsigned char presentation_ind;
unsigned char screening_ind;
unsigned char digits [IISDN_MAX_DIGITS+2];
```

Input

num_digits

Number of Digits. Specifies the number of digits in the Calling Party Number. A value of 0x0000 indicates the Calling Party number is not available and the remainder of this structure can be omitted. The application determines valid Calling Party numbers by the current network dialing plan. This number must be less than IISDN_MAX_DIGITS.

num_type

Number Type. Corresponds to the Calling Party Number Types as ignored by ITU-T. This field is usually specified as IISDNnumtNATIONAL (0x02). Possible values include:

IISDNnumtUNKNOWN 0x00

Unknown number type, used when there is no knowledge of the type of number.

IISDNnumtINTERNATIONAL 0x01

International number.

IISDNnumtNATIONAL 0x02

National number.

IISDNnumtSUBSCRIBER 0x04

subscriber number.

IISDNnumtABBREVIATED 0x06

Abbreviated number.

num_plan

Numbering Plan Identification. Corresponds to Calling Party numbering plan identification as defined by ITU-T. This field is usually specified as IISDNnumpISDN (0x01). Possible values include:

IISDNnumpUNKNOWN 0x00

Unknown Numbering Plan.

IISDNnumpISDN 0x01

Numbering per ISDN Recommendation E.164/E.163.

IISDNnumpTELEPHONY 0x02

Standard telephone numbering plan.

IISDNnumpPRIVATE 0x09

Private numbering plan.

presentation_ind

Presentation Indicator. Identifies whether the calling user wants the Calling Party number presented to the called user. If no preference, specify a value of 0x00. Possible values include:

IISDNpresALLOWED	0x00
Presentation allowed or no preference.	
IISDNpresRESTRICTED	0x01
Presentation restricted/not allowed.	
IISDNpresNUM_NOT_AVAIL	0x02
Calling Party number not available due to inter-networking.	

screening_ind

Screening Indicator. Identifies whether screening has been performed on the calling user and the status of any screening performed. If no preference, specify a value of 0x00. Possible values include:

IISDNscrUSER_NOT_SCREENED	0x00
User provided, not screened.	
IISDNscrUSER_VER_PASSED	0x01
User provided, verified and passed.	
IISDNscrUSER_VER_FAILED	0x02
User provided, verified and failed.	
IISDNscrNETWORK_PROVIDED	0x03
Network provided.	

digits

Digits. Calling Party number in ASCII digits. Number of digits in this string must equal the number of digits specified in *num_digits*. Eight-bit values in this field must be between 0x30 and 0x39 (ASCII 0 through 9).

Cause Data (IISDN_CAUSE)

Description

Cause Data provides information relating to the source and reason for generation of an ISDN message. This structure contains the following components:

- Coding standard being used
- Location of the equipment generating the message
- Additional diagnostic information

The Cause Data structure detailed below is used in the following Call Control messages:

- *L4L3mCLEAR_REQUEST*
- *L4L3mINFO_REQUEST*
- *L3L4mCLEAR_REQUEST*
- *L3L4mDISCONNECT*

Note: Using a non-default cause value in an *L4L3mCALL_REQUEST* message may cause your application to fail ISDN conformance testing.

Input Fields

```
unsigned char coding_standard;
unsigned char location;
unsigned char cause_val;
unsigned char diag_len;
unsigned char diags;
```

Input

coding_standard

Coding Standard. Indicates the coding standard used to construct this message. The default value of IISDNcodCCITT (0x00) should be used on most cases. Possible values include:

IISDNcodCCITT 0x00

ITU-T coding standard used for this message. This value should be used unless the progress indication cannot be represented using standard ITU-T coding.

IISDNcodNATIONAL_STD 0x02

National standard coding values not supported by ITU-T coding values used for this message. Recipient of this message should be capable of interpreting this meaning.

IISDNcodSTD_SPF_2_LOC 0x03

Coding standard in use is specific to the location.

location

Location. Indicates the location of the user for the generated message. Possible values include:

IISDNlocUSER 0x00

User.

IISDNlocPVT_LOCAL 0x01

Private network serving the local user.

IISDNlocPUB_LOCAL 0x02

Public network serving the local user.

IISDNlocTRANSIT_NET 0x03

Transit network.

IISDNlocPUB_REMOTE 0x04

Public network serving the remote user.

IISDNlocPVT_REMOTE 0x05

Private network serving the remote user.

IISDNlocINTERNATIONAL 0x07

International network.

IISDNlocBEY_INTERWORK 0x10

Network beyond the interworking point.

cause_val

Cause. Indicates the reason the message in which this structure is contained was generated. A value of 0x00 indicates either the default cause is used or the cause information is unavailable. Refer to *Volume 6, Appendix D, BSMI Cause Values,* for a listing of possible values.

diag_len

Diagnostics Length. Specifies the number of diagnostic information bytes that follow this char. If this value is 0x00, no diagnostic information is available or required. Not all cause values support additional diagnostic information. Diagnostics are encoded only if *cause_val* is a non-zero value.

diags

Diagnostic codes in hex bytes. Specifies the number of bytes included in this array in *diag_len*. Not all cause values support additional diagnostic information.

Refer to *Volume 6, Appendix D, BSMI Cause Codes* for more information on cause values.

Connected Address (IISDN_CONNECTED_ADDRESS)

Description

Connected Address provides information necessary to identify the actual party connected to the call and contains the following components:

- Number of digits in the originating telephone number (or zero if the number is not available)
- Type of number (usually National for domestic U.S. calls)
- Numbering plan used (typically ISDN)
- Originating telephone number, if available

The Connected Address structure detailed below is used in the following messages:

- *L4L3mALERTING_REQUEST*
- *L4L3mCLEAR_REQUEST*
- *L4L3mCONNECT_REQUEST*
- *L3L4mALERTING*
- *L3L4mCLEAR_REQUEST*
- *L3L4mCONNECT*
- *L3L4mDISCONNECT*

Input Fields

```
unsigned short num_digits;
unsigned char num_type;
unsigned char num_plan;
unsigned char digits [IISDN_MAX_DIGITS];
```

Input

num_digits

Number of Digits. Specifies the number of digits in the Connected Address Number. A value of 0x0000 indicates the Connected Address number is not available and the remainder of this structure can be ignored. The application determines valid Connected Address numbers using the current network dialing plan. This number must be less than 0x18.

num_type

Number Type. Corresponds to the Connected Address Number Types as defined by ITU-T. Possible values include:

IISDNnumtUNKNOWN 0x00

Unknown number type, used when there is no knowledge of the type of number.

IISDNnumtINTERNATIONAL 0x01

International number.

IISDNnumtNATIONAL 0x02

National number.

IISDNnumtSUBSCRIBER 0x04

Subscriber number.

IISDNnumtABBREVIATED 0x06

Abbreviated number.

num_plan

Numbering Plan Identification. Corresponds to Connected Address numbering plan identification as defined by ITU-T. This field is usually specified as IISDNnumpISDN (0x01). Possible values include:

IISDNnumpUNKNOWN 0x00

Unknown Numbering Plan.

IISDNnumpISDN 0x01

Numbering per ISDN Recommendation E.164/E.163.

IISDNnumpTELEPHONY 0x02

Standard telephone numbering plan.

IISDNnumpPRIVATE 0x09

Private numbering plan.

digits

Digits. Connected Address number in ASCII digits. Number of digits in this string must equal the number of digits specified in *num_digits*. Eight-bit values in this field must be between 0x30 and 0x39 (ASCII 0 through 9).

Information Element (IISDN_IE_STRUCT)

Description

Information Element (IE) allows the application to append IEs to messages generated by the Dialogic® Brooktrout® firmware. The application can use this feature to add optional or site-specific IEs to a standard ISDN message. When used with the *L4L3mUNIVERSAL* BSMI message, the Information Element structure permits the application to issue non-standard ISDN messages. This structure contains the following components:

- Information Element identifier
- Length of the IE
- Application-specific information (data bytes)

Up to 30 IEs can be transmitted in a single ISDN message. This number includes those IEs generated automatically by the Dialogic® Brooktrout® firmware.

Single octet IEs are specified as follows:

- Information Element identifier
- Length = 0
- No additional data bytes

Multiple IEs are specified as a string of bytes that include the above components. IEs must be specified in ascending order within a codeset. Only locking codeset shifts are allowed.

Use the same structure also to specify a locking codeset shift to be applied to all subsequent IEs in the message. Once the firmware shifts to an alternate codeset, only shifts to higher codesets are allowed in the same message. When making a codeset shift, the components of the structure are specified as follows:

- Codeset shift IE in place of the IE identifier
- Length = 0x00 (no data bytes to follow)
- No data bytes are included in this structure (the byte that follows is the first octet of the next IE)

The total message size, including the main body of the message and all IE structures, must be less than or equal to the L4L3m buffer size.

The Information Element structure is used in the following messages:

- *L4L3mALERTING_REQUEST*
- *L4L3mCALL_PROCEEDING_REQUEST*
- *L4L3mCALL_REQUEST*
- *L4L3mCALL_REQUEST*
- *L4L3mFACILITY_REQUEST*
- *L4L3mINFO_REQUEST*
- *L4L3mPROGRESS_REQUEST*
- *L4L3mSETUP_ACK_REQUEST*
- *L4L3mUNIVERSAL*

With the exception of the *L4L3mUNIVERSAL* message, Dialogic® Brooktrout® firmware ensures the proper IE ordering within a message. When using *L4L3mUNIVERSAL* the host application is responsible for proper IE ordering.

Input Fields

```
unsigned char ie_id;
unsigned char ie_length;
unsigned char ie_data [1];
```

Input

ie_id

IE Identifier. First octet of the IE. The receiving application must be able to determine the IE from this information.

ie_length

Length. Specifies the number of octets in this IE.

ie_data

Data Bytes. Variable length.

Note: The IE structure and the message containing it must fit in the size of *L3_to_L4_struct* or *L4_to_L3_struct*. Dialogic® Brooktrout® firmware call control messages are always limited to 512 bytes.

Progress Indication (IISDN_PROGRESS)

Description

The Progress Indication notifies the user of the current status that the B-channel and interface is in for the certain call. Its structure is designed to allow the user information on the location of the call as well as its description. During initial SETUP, Progress Indication feeds the user with the feature availability and the bearer selection of the call and notifies when the sending message has been completed.

The Progress Indication structure detailed below is used in the following messages:

- *L4L3mALERTING_REQUEST*
- *L4L3mCALL_PROCEEDING_REQUEST*
- *L4L3mCLEAR_REQUEST*
- *L4L3mCONNECT_REQUEST*
- *L4L3mSETUP_ACK_REQUEST*
- *L3L4mALERTING*
- *L3L4mCALL_PROCEEDING*
- *L3L4mCLEAR_REQUEST*
- *L3L4mCONNECT*
- *L3L4mDISCONNECT*
- *L3L4mPROGRESS*

Input Fields

```
unsigned char coding_standard;
unsigned char location;
unsigned char PROGRESS_DSCR;
IISDN_CAUSE cause;
IISDN_USER_INFO user_info;
unsigned char ie_count;
IISDN_CALL_ID call_id;
IISDN_IE_STRUCT ie;
```

Input

coding_standard

Coding Standard. Indicates the coding standard used to construct this message. Use the default value of IISDNcodCCITT (0x00) for most cases. Possible values include:

IISDNcodCCITT 0x00

ITU-T coding standard used for this message. Do not use this value unless the progress indication cannot be represented using standard ITU-T coding.

IISDNcodNATIONAL_STD 0x02

National standard coding values not supported by ITU-T coding values used for this message. Recipient of this message should be capable of interpreting this meaning.

IISDNcodSTD_SPF_2_LOC 0x03

Coding standard used is specific to the location to which the message is sent.

location

Location. Indicates the location of the user that generated the message. Possible values include:

IISDNlocUSER 0x00

User.

IISDNlocPVT_LOCAL 0x01

Private network serving the local user.

IISDNlocPUB_LOCAL 0x02

Public network serving the local user.

IISDNlocTRANSIT_NET 0x03

Transit network.

IISDNlocPUB_REMOTE 0x04

Public network serving the remote user.

IISDNlocPVT_REMOTE 0x05

Private network serving the remote user.

IISDNlocINTERNATIONAL 0x07

International network.

IISDNlocBEY_INTERWORK 0x10

Network beyond the interworking point.

PROGRESS_DSCR

Indicates the progress of the call.

IISDNprogUNKNOWN 0x00

Information not available; default value.

IISDNprogNOT_ISDN_INBAND 0x01

Call is not end-to-end ISDN; additional information for this call might be available in-band. Use this selection to indicate to the destination processor that digits or other in-band signaling might be present and should be monitored.

IISDNprogDEST_NOT ISDN 0x02

Call destination (called party) is not ISDN.

IISDNprogORIG_NOT_ISDN 0x03

Call origination (calling party) is not ISDN.

IISDNprogRETURNED_ISDN 0x04

Call has been returned to the ISDN.

IISDNprogINBAND_INFO_AVL 0x08

Additional in-band information for this call is available in-band. This selection indicates that digits or other in-band signaling is present and should be monitored.

cause

See *Cause Data (IISDN_CAUSE) on page 860*.

user_info

See *User Info (IISDN_USER_INFO) on page 875*.

ie_count

Counts the number of IEs in this message.

call_id

See *Call ID (IISDN_CALL_ID) on page 854*.

ie

See *Information Element (IISDN_IE_STRUCT) on page 865*.

Q.933 DLCI Negotiation (IISDN_Q922_DLCI)

Description

Q.933 DLCI Negotiation is used to perform DLCI negotiation similar to B-channel negotiation in Q.931 applications. The application uses this structure to populate (outgoing calls) and decode (incoming calls) a data link connection identifier IE in SETUP messages. The structure indicates the DLCI selection setting (preferred or exclusive) and the DLCI requested (for outgoing calls) or assigned (for incoming calls).

The Q.933 DLCI Negotiation structure detailed below is used in the following messages:

- *L4L3mALERTING_REQUEST*
- *L4L3mCALL_PROCEEDING_REQUEST*
- *L4L3mCALL_REQUEST*
- *L4L3mCONNECT_REQUEST*
- *L3L4mALERTING*
- *L3L4mCONNECT*
- *L3L4mSETUP_IND*

Input Fields

```
unsigned char dlci_present;
unsigned char preferred;
```

Input

dlci_present

DLCI Present. Specifies whether the data structure supplies a DLCI value. Possible values are:

0x00 = DLCI information not available
0x01 = DLCI provided

preferred

Preferred DLCI. Specifies whether the DLCI included in the data structure is preferred or exclusive.

Preferred DLCI	DLCI allows the negotiation of the value between the network and the module. Value is 0x01.
Exclusive DLCI	Must use a DLCI value. No negotiation is allowed. Value is 0x00.

Redirecting Number (IISDN_REDIRECT_NUM)

Description

Redirecting Number provides information necessary to construct the Redirecting Number IE. This IE is used to indicate the reason and source of a forwarded call and contains the following components:

- Number of digits in the Redirection (new destination) telephone number
- Type of number (usually National for North American numbers).
- Numbering plan used (typically ISDN)
- Reason the call is being redirected
- Redirection (new destination) telephone number

The Redirecting Number structure detailed below is used in the following messages:

- *L4L3mALERTING_REQUEST*
- *L4L3mCALL_REQUEST*
- *L4L3mCONNECT_REQUEST*
- *L3L4mALERTING*
- *L3L4mCONNECT*
- *L3L4mSETUP_IND*

Input Fields

```
unsigned short num_digits;
unsigned char num_type;
unsigned char num_plan;
unsigned char presentation_ind;
unsigned char screening_ind;
unsigned char redir_reason;
unsigned char digits [IISDN_MAX_DIGITS];
```

Input

num_digits

Number of Digits. Specifies the number of digits in the Redirection number. The application determines valid Redirection Party numbers by the current network dialing plan. A value of zero indicates this information is not available. This number must be less than 0x18.

num_type

Number Type. Corresponds to the Calling Party Number Types as defined by ITU-T. Possible values include:

IISDNnumtUNKNOWN 0x00

Unknown number type, used when there is no knowledge of the type of number.

IISDNnumtINTERNATIONAL 0x01

International number.

IISDNnumtNATIONAL 0x02

National number.

IISDNnumtSUBSCRIBER 0x04

Subscriber number.

IISDNnumtABBREVIATED 0x06

Abbreviated number.

num_plan

Numbering Plan Identification. Corresponds to Calling Party numbering plan identification as defined by AT&T. The firmware usually specifies as IISDNnumpISDN (0x01). Possible values include:

IISDNnumpUNKNOWN 0x00

Unknown Numbering Plan.

IISDNnumpISDN 0x01

Numbering per ISDN Recommendation E.164/E.163.

IISDNnumpTELEPHONY 0x02

Standard telephone numbering plan.

IISDNnumpPRIVATE 0x05

Private numbering plan.

presentation_ind

Presentation Indicator. Identifies whether the calling user wants the Calling Party number presented to the called user. If no preference, specify a value of 0x00. Possible values include:

IISDNpresALLOWED 0x00

Presentation allowed or no preference.

IISDNpresRESTRICTED 0x01

Presentation restricted/not allowed.

IISDNpresNUM_NOT_AVAIL 0x02

Calling Party number not available due to inter-networking.

screening_ind

Screening Indicator. Identifies whether screening has been performed on the calling user and the status of any screening performed. If no preference, specify a value of 0x00. Possible values include:

IISDNscrUSER_NOT_SCREENED 0x00

User provided, not screened.

IISDNscrUSER_VER_PASSED 0x01

User provided, verified and passed.

IISDNscrUSER_VER_FAILED 0x02

User provided, verified and failed.

IISDNscrNETWORK_PROVIDED 0x03

Network provided.

redir_reason

Redirecting Reason. Indicates the reason the call has been redirected. Possible values include:

IISDNrrsnUNKNOWN	0x00
Unknown reason for redirection.	
IISDNrrsnCALL_FWD_BUSY	0x01
Original destination party is busy.	
IISDNrrsnCALL_FWD_NOANS	0x02
Original destination number has not answered.	
IISDNrrsnOOS	0x05
Original destination number is out of service.	
IISDNrrsnCALL_FWD_DTE	0x0A
The called data terminal equipment provided redirect instructions.	
IISDNrrsnCALL_FWD_ALL	0x0F
Forwards all calls to original destination number.	

digits

Digits. Redirection number (new destination) in ASCII digits. Number of digits in this string must equal the number of digits specified in *num_digits*. Eight-bit values in this field must be between 0x30 and 0x39 (ASCII 0 through 9).

User Info (IISDN_USER_INFO)

Description

User Info allows the passing of up to 130 bytes of information over the network without establishing a switched end-to-end connection. This structure provides the information necessary to construct the User-user IE. This IE contains the following components:

- Number of bytes of information being transferred
- User-specific information

The User Info structure detailed below is used in the following messages:

- *L4L3mALERTING_REQUEST*
- *L4L3mCALL_REQUEST*
- *L4L3mCLEAR_REQUEST*
- *L4L3mCONNECT_REQUEST*
- *L4L3mPROGRESS_REQUEST*
- *L4L3mUSER_INFO*
- *L4L3mALERTING_REQUEST*
- *L3L4mCLEAR_REQUEST*
- *L3L4mCONNECT*
- *L3L4mDISCONNECT*
- *L3L4mPROGRESS*
- *L3L4mSETUP_IND*
- *L3L4mUSER_INFO*

Input Fields

```
unsigned short len;
unsigned char info [IISDN_MAX_USER_INFO];
```

Input

len

Length. Specifies the number of bytes that follow this char. If this value is 0x00, no information bytes follow this short integer. Possible values are between 0 and 130 (decimal).

info

User info, the number of bytes specified by *len*. Possible values up to IISDN_MAX_USER_INFO.

24 - R2 Signaling Protocol with BSMI

This chapter describes the subset of ISDN messages and events used in the BSMI implementation of an R2 signaling protocol.

This chapter has the following sections:

- *Application to Stack (Host to Module) Messages*
- *R2 Signaling L4L3 Messages*
- *Stack to Application (Module to Host) Messages*
- *Normal Event Sequence*
- *R2 Signaling L3L4 Messages*

Application to Stack (Host to Module) Messages

R2 signaling for Host to Module message summaries are shown in:

- *Table 12, R2 Signaling Management Messages (L4L3)*
- *Table 13, R2 Signaling Call Control Messages (L4L3)*

Table 12. R2 Signaling Management Messages (L4L3)

Message	Meaning
L4L3mDISABLE_CAS ID: 0xC7	Disables the R2 protocol. This message is for any timeslot on an E1 span.
L4L3mENABLE_CAS ID: 0xC6	Starts the R2 protocol. This message is issued for each timeslot on the E1 span.
L4L3mREQ_ABCD_DATA ID: 0xBC	Requests the module to return to the host all signaling bits from all B-channels in module.
L4L3mSET_CAS_SIGNALING_BITS	Sets the signaling bit pattern on a particular B-channel.

Table 13 lists call control related messages. Use the call control messages to:

- Place an outbound call.
- Answer an inbound call.
- Disconnect a call or reject and incoming call.

Table 13. R2 Signaling Call Control Messages (L4L3)

Message	Meaning
L4L3mALERTING_REQUEST ID: 0x83	Informs the network an incoming call is ringing (send ALERTING message to the network).
L4L3mCALL_PROCEEDING_REQUEST ID: 0x89	Notifies the switching equipment originating the call that the called party's line is free and being alerted (ringing).

Table 13. R2 Signaling Call Control Messages (L4L3) (Continued)

Message	Meaning
L4L3mCALL_REQUEST ID: 0x81	Places an outbound call.
L4L3mCAS_CHAN_BLOCK ID: 0xD6	Asserts blocking pattern on the line. Only valid from the idle state. Transitions the B-channel to the blocking state.
L4L3mCAS_CHAN_UNBLOCK ID: 0xD7	Asserts idle pattern on the line and returns to the idle state. Only valid from the blocking state.
L4L3mCLEAR_REQUEST ID: 0x85	Disconnects a connected call or rejects an incoming call.
L4L3mCOLLECT_DIGITS ID: 0xC9	Acknowledges an incoming R2 MF digit and request the next one.
L4L3mCONNECT_REQUEST ID: 0x84	Answers an incoming call.
L4L3mINFO_REQUEST ID: 0x8B	Dials a test R2 MF string of tones.

Numbering Conventions

The R2 numbering conventions follow the ISDN conventions. E1 trunks are numbered starting from 0 and B-channels are logically numbered starting at 0. Consequently, the valid range of B-channels for a given E1 trunk is from 0 to 29. These values correspond to the physical timeslots 1 to 15 and 17 to 31, inclusive.

Arguments

Arguments are passed to the R2 protocol using the standard ISDN Layer 4 to Layer 3 (**`L4_to_L3_struct`**) data structure.

The only message using a call reference number is the clear request message. Although ISDN allows the application to specify its own call reference number for outgoing calls, R2 signaling does not allow this functionality. When clearing a call, the call reference number is the one obtained from previous messages associated with the call.

R2 Signaling L4L3 Messages

The following message subset is used in the R2 signaling protocol for Host to Module messaging. Specific message details begin on *page 885*:

- *L4L3mALERTING_REQUEST on page 881*
- *L4L3mCALL_PROCEEDING_REQUEST on page 883*
- *L4L3mCALL_REQUEST on page 885*
- *L4L3mCAS_CHAN_BLOCK on page 888*
- *L4L3mCAS_CHAN_UNBLOCK on page 890*
- *L4L3mCLEAR_REQUEST on page 892*
- *L4L3mCOLLECT_DIGITS on page 894*
- *L4L3mCONNECT_REQUEST on page 896*
- *L4L3mDISABLE_CAS on page 898*
- *L4L3mENABLE_CAS on page 900*
- *L4L3mINFO_REQUEST on page 903*
- *L4L3mSET_CAS_SIGNALING_BITS on page 906*

L4L3mALERTING_REQUEST

Description

Accepts an incoming call reported by the module through *L3L4mSETUP_IND* but does not answer it. The module notifies the network that the called subscriber is free and its terminal (telephone) is ringing (the module automatically generates the ringback tone). The application can later decide to answer the call (*L4L3mCONNECT_REQUEST*) or not, in which case the caller receives a ringback tone until it decides to terminate the call.

Arguments

The trunk must always be specified in **`L4_to_L3_struct`**`.lapdid`.

The B-channel is determined by setting **`L4_to_L3_struct`**`.call_ref` to: `((Trunk <<8) | Bchannel)`.

Expected Response

None.

Data Checking

The timeslot must be in the range 0 through 29.

The protocol must be initialized for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in an inbound seized state (an incoming call is detected but not answered by the host), with call setup information (R2 MF compelled signaling) still in progress.

Error Codes

L3L4errCALL_REF_ERROR	B-channel values obtained from *L4_to_L3_struct.data.al_con_data.bchannel* and from *L4_to_L3_struct.call_ref* did not match.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in *L4_to_L3_struct.data.al_con_data* structure.

L3L4errINVALID_MSG_FOR_STATE	Message sent while the protocol was in a state that did not accept it. *L4L3mALERTING_REQUEST* is only accepted when the incoming call reported to the application but the application has not yet answered.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks are present on the module.

See Also

None.

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mALERTING_REQUEST;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.call_ref = ((L4L3m.lapdid << 8) | 20);
BsmiControlWrite (fd, &L4L3);
```

L4L3mCALL_PROCEEDING_REQUEST

Description

Notifies the module that the call address (called party number) is complete and the R2 MF signaling proceeds to group B. Use this message if the host requests (see option `R2_COLLECT_DIGITS` in *R2Options* field of *IISDN_E1_CAS_R2_DATA*) notification of each incoming digit so it can control the flow of the R2 MF signaling.

Arguments

The trunk must always be specified in **`L4_to_L3_struct`**`.lapdid`.

The B-channel is determined by setting **`L4_to_L3_struct`**`.call_ref` to: `((Trunk <<8) | Bchannel)`.

Expected Response

None.

Data Checking

The timeslot must be in the range 0 through 29.

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in an inbound seized state (an incoming call is detected, but not answered by the host), with call setup information (R2 MF compelled signaling) still in progress.

Initialize the protocol (*L4L3mENABLE_CAS*) with option R2_COLLECT_DIGITS in *R2Options* field of *IISDN_E1_CAS_R2_DATA*.

Error Codes

L3L4errCALL_REF_ERROR	B-channel values obtained from *L4_to_L3_struct.data.proc_data.bchannel* and from *L4_to_L3_struct.call_ref* did not match.
L3L4errFEATURE_NOT_ACTIVE	The channel is not configured to let the application handle the R2 MF signaling (channel must be configured with option R2_COLLECT_DIGITS set in: *L4_to_L3_struct.data.data.cas_data.cas_params.e1_cas_r2_data.R2Options*
L3L4errINVALID_B_CHANNEL	B-channel value exceeded maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in structure *L4_to_L3_struct.data.proc_data.*
L3L4errINVALID_MSG_FOR_STATE	The message was sent while the protocol was in a state that did not accept it. The firmware only accepts *L4L3mALERTING_REQUEST* when an incoming call reported to the application but the application has not yet answered and the R2 MF signaling is in progress and collecting the address (called party) digits.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

See Also

IISDN_E1_CAS_R2_DATA

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mCALL_PROCEEDING_REQUEST;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.call_ref = ((L4L3m.lapdid << 8) | 20);
BsmiControlWrite (fd, &L4L3);
```

L4L3mCALL_REQUEST

Description

This message is used to start an outgoing call.

The called party number and the calling party number are supplied in the *Called Party (IISDN_CALLED_PARTY)* and *Calling Party (IISDN_CALLING_PARTY)* structures respectively.

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.`lapdid`
- **L4_to_L3_struct**.`data.call_req_data.bchannel`
- Set **L4_to_L3_struct**.`call_ref` to:
  `(Trunk <<8) | Bchannel)`
- **L4_to_L3_struct**.`data.call_req_data` must be set properly.

Expected Response

The protocol stack sends no response message if the outbound call is successfully started. Otherwise, error messages are sent.

Data Checking

The trunk and B-channel numbers are verified. The range of valid B-channel depends on the hardware being utilized:

- T1 trunks have 24 B-channels
- E1 trunks have 30 B-channels
- Analog Loop Start modules have variable number of trunks

The protocol must be initialized for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in the idle state.

Digits specified as Called Party number and Calling Party number are checked for validity.

Error Codes

L3L4errGLARE	The firmware received a signal indicating the presence of an inbound call when the module was trying to initiate an outbound call.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in structure *L4_to_L3_struct.data.call_req_data. call_party_num_digits* must have a positive value.
L3L4errINVALID_MSG_FOR_STATE	Message sent while the protocol was in a state that did not accept it. *L4L3mCALL_REQUEST* is only accepted when line is idle.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

See Also

Called Party (IISDN_CALLED_PARTY), *Calling Party (IISDN_CALLING_PARTY)*, *IISDN_CALL_REQ_DATA*

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mCALL_REQUEST;

//This example uses trunk 0, channel 20
L4L3.lapdid = 0;
L4L3.data.call_req_data.bchannel = 20;

//Called party number.
L4L3.data.call_req_data.called_party.num_digits = 10;
L4L3.data.call_req_data.called_party.num_type = IISDNnumtUNKNOWN;
L4L3.data.call_req_data.called_party.num_plan = IISDNnumpUNKNOWN;
strncpy( &(L4L3.data.call_req_data.called_party.digits[0]),
   "1234567890", 10);

//Calling party number (this is not required by the protocol)
L4L3.data.call_req_data.calling_party.num_digits = 10;
L4L3.data.call_req_data.calling_party.num_type = IISDNnumtUNKNOWN;
L4L3.data.call_req_data.calling_party.num_plan = IISDNnumpUNKNOWN;
strncpy( &(L4L3.data.call_req_data.calling_party.digits[0]),
   "0987654321", 20);

BsmiControlWrite(fd, &L4L3);
```

L4L3mCAS_CHAN_BLOCK

Description

This message blocks a B-channel, thus preventing it from receiving inbound calls. The B-channel remains blocked until *L4L3mCAS_CHAN_UNBLOCK* is received. You cannot initiate outbound calls while the B-channel is blocked.

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.lapdid
- **L4_to_L3_struct**.data.cas_data.bchannel
- Set **L4_to_L3_struct**.call_ref to:
  ((Trunk <<8) | Bchannel)

Expected Response

The module responds with *L4L3mCAS_CHAN_BLOCK*, and **L3_to_L4_struct**.data.b_channel_status is set to R2_BLOCKED_NEAR_END_BLOCKING.

Data Checking

The timeslot must be in the range 0 through 29, the protocol must be in the idle state for this B-channel, and the B-channel must be active (the protocol is started).

Error Codes

L3L4errINVALID_B_CHANNEL	B-channel value exceeded maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in structure *L4_to_L3_struct.data*.
L3L4errINVALID_MSG_FOR_STATE	Message sent while the protocol was in a state that did not accept it. *L4L3mCALL_PROCEEDING_REQUEST* is only accepted when line is idle.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

See Also

IISDN_BCHANNEL_ID

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mCAS_CHAN_BLOCK;
L4L3.data.cas_data.bchannel = 20;
L4L3.lapdid  = 1;
BsmiControlWrite(fd, &L4L3);
```

L4L3mCAS_CHAN_UNBLOCK

Description

This message unblocks a B-channel, allowing it to start outbound calls and receive inbound calls.

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.lapdid
- **L4_to_L3_struct**.data.cas_data.bchannel
- Set **L4_to_L3_struct**.call_ref to:
  ((Trunk <<8) | Bchannel)

Expected Response

The module responds with *L4L3mCAS_CHAN_BLOCK*, and the application sets *L3_to_L4_struct.data.b_channel_status* to R2_BLOCKED_NEAR_END_BLOCKING.

Data Checking

The timeslot must be in the range 0 through 29, and the protocol must be in the blocked state for this B-channel. The B-channel must be active (the protocol is started).

Error Codes

L3L4errINVALID_B_CHANNEL	B-channel value exceeded maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in *L4_to_L3_struct.data* structure.
L3L4errINVALID_MSG_FOR_STATE	Message sent while the protocol was in a state that did not accept it. *L4L3mCALL_PROCEEDING_REQUEST* is only accepted when line is blocked by a previous *L4L3mCAS_CHAN_BLOCK*.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

See Also

IISDN_BCHANNEL_ID

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mCAS_CHAN_UNBLOCK;
L4L3.data.cas_data.bchannel = 20;
L4L3.lapdid  = 1;
BsmiControlWrite(fd, &L4L3);
```

L4L3mCLEAR_REQUEST

Description

This message clears (tears down) a call or refuses an incoming call. If an incoming call is refused, use *L4L3mCLEAR_REQUEST* after notification by *L3L4mSETUP_IND* (instead of accepting the message with *L4L3mALERTING_REQUEST* or *L4L3mCONNECT_REQUEST*).

If using *L4L3mCLEAR_REQUEST* to clear an incoming call, then the reason must be supplied in **L4L3**.`data.clr_data.r2_call_status.group_B`.

The possible values are specified in *iisdh.h* include:

- IISDN_R2MFCP_LINE_BUSY
- IISDN_R2MFCP_LINE_OUT_OF_ORDER
- IISDN_R2MFCP_LINE_UNALLOCATED

Arguments

The trunk must always be specified in **L4_to_L3_struct**.`lapdid`.

The B-channel is determined by setting **L4_to_L3_struct**.`call_ref` to: `((Trunk <<8) | Bchannel)`.

Expected Response

None.

Data Checking

The channel and trunk as encoded in the call reference number is checked.

Error Codes

L3L4errINVALID_B_CHANNEL	B-channel value exceeded maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in *L4_to_L3_struct.data* structure.
L3L4errINVALID_MSG_FOR_STATE	Message sent while the protocol was in a state that did not accept it. The application only accepts *L4L3mCLEAR_REQUEST* when a call is in progress (inbound or outbound). When *L3L4mSETUP_IND* is sent to the host inbound calls are cleared. Consequently, you cannot terminate an inbound call that is still characterized (from the moment the line is seized and *L3L4mPRE_SEIZE* can be sent to the host, to the moment all necessary data is available and the module sends *L3L4mSETUP_IND* to the host).
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks are present on the module.

See Also

IISDN_CLR_DATA

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mCLEAR_REQUEST;

L4L3.call_ref = CRN; // from a prior event

L4L3.data.clr_data.r2_call_status.group_B =
(unsignedchar)IISDN_R2MFCP_LINE_BUSY;
```

L4L3mCOLLECT_DIGITS

Description

Notifies the module that the current R2 MF digit is acknowledged. The module sends the address (called party) digits received thus far through *L3L4mINFO* to the host and sends a tone to the network indicating it needs to send the next address digit.

Arguments

The trunk must always be specified in **`L4_to_L3_struct`**`.lapdid`.

The B-channel is determined by setting **`L4_to_L3_struct`**`.call_ref` to: `((Trunk <<8) | Bchannel)`.

Expected Response

L3L4mINFO returns none.

Data Checking

The timeslot must be in the range 0 through 29.

The protocol must be initialized for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in an inbound seized state (an incoming call is detected, but not answered by the host), with call setup information (R2 MF compelled signaling) still in progress.

L4_to_L3_struct.dial_data.num_digits must be greater than 0.

The protocol must be initialized (*L4L3mENABLE_CAS*) with option R2_COLLECT_DIGITS in *R2Options* field of structure IISDN_E1_CAS_R2_DATA set.

Error Codes

L3L4errCALL_REF_ERROR	B-channel values obtained from *L4_to_L3_struct.data.dial_data.bchannel* and from *L4_to_L3_struct.call_ref* did not match.
L3L4errFEATURE_NOT_ACTIVE	The channel is not configured to let the application handle the R2 MF signaling (Configure the channel with option R2_COLLECT_DIGITS set in: *L4_to_L3_struct.data.data.cas_data. cas_params.e1_cas_r2_data.R2Options).*
L3L4errINVALID_B_CHANNEL	B-channel value exceeded maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in structure *L4_to_L3_struct.data.dial_data.*
L3L4errINVALID_MSG_FOR_STATE	Message sent while the protocol was in a state that did not accept it. *L4L3mCOLLECT_DIGITS* is only accepted when: ■ The incoming call is present ■ R2 MF signaling is active ■ The originating end is ready to sent the address (called party) digits.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks are present on the module.

See Also

IISDN_E1_CAS_R2_DATA, IISDN_DIAL_DATA

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mCOLLECT_DIGITS;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.call_ref = ((L4L3m.lapdid << 8) | 20);
BsmiControlWrite (fd, &L4L3);
```

L4L3mCONNECT_REQUEST

Description

This message is used to answer an incoming call.

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.lapdid
- **L4_to_L3_struct**.data.al_con_data.bchannel
- Set **L4_to_L3_struct**.call_ref to:
  ((Trunk <<8) | Bchannel)
- **L4_to_L3_struct**.data.al_con_data must be set properly.

Expected Response

After the connection completes, *L3L4mCONN_ACK_IND* is sent to the host.

Data Checking

The E1 trunk number and the channel are checked.

Error Codes

L3L4errCALL_REF_ERROR	B-channel values obtained from *L4_to_L3_struct.data.al_con_data.bchannel* and from *L4_to_L3_struct.call_ref* did not match.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in the *L4_to_L3_struct.data.al_con_data* structure.
L3L4errINVALID_MSG_FOR_STATE	Message sent while the protocol was in a state that did not accept it. *L4L3mCONNECT_REQUEST* is only accepted when the inbound call is present and fully characterized (the complete address (called party number) is available). You can call *L4L3mCONNECT_REQUEST* only once for a particular inbound call.

L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks are present on the module.

See Also

Alerting and Connecting Data Message (IISDN_AL_CON_DATA)

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mCONNECT_REQUEST;

L4L3.lapdid = 0; // Trunk number
L4L3.data.al_con_data.channel = 20; // B-channel
L4L3.call_ref = CRN;   // from prior L3L4mSETUP_IND event

// This answer signal is recommended.

L4L3.data.al_con_data.r2_call_status.group_B =
    (unsigned c2.har )IISDN_R2MFCP_LINE_FREE_CHARGE;
// Number of ring cycles before answering.

L4L3.data.al_con_data.r2_call_status.NumberRings =
       (unsigned char )1;
BsmiControlWrite(fd, &L4L3);
```

L4L3mDISABLE_CAS

Description

This message stops the protocol running on the specified line. Any ongoing calls are automatically terminated.

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.lapdid
- **L4_to_L3_struct**.data.cas_data.bchannel
- **L4_to_L3_struct**.call_ref set to:
  ((Trunk <<8) | Bchannel)
- **L4_to_L3_struct**.data.cas_data must be set properly.

Expected Response

The module responds with *L3L4mCAS_STATUS* and **L3_to_L4_ struct**.data.b_channel_status is set to IISDNbcsOUT_OF_SERVICE.

Data Checking

The value of the timeslot is checked along with whether or not the protocol is active.

Error Codes

L3L4errINVALID_B_CHANNEL	B-channel value exceeded maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks are present on the module.

See Also

IISDN_BCHANNEL_ID

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mDISABLE_CAS;

L4L3.data.cas_data. = 15;//same channel as ENABLE example
L4L3.lapdid  = 0;       // same trunk as in ENABLE example
BsmiControlWrite(fd, &L4L3);
```

L4L3mENABLE_CAS

Description

This message specifies the programming of the R2 protocol and provides the data that differentiates the R2 signaling of one country from another.

The *L4L3mENABLE_CAS* message must pass the following arguments:

- E1 trunk number
- B-channel
- Signaling type

The structure elements and corresponding values that specify these arguments are shown below:

L4L3.data.cas_data.bchannel	0 to 29
L4L3.lapdid	E1 trunk number (0 or 1)

The R2 protocol has *Include* files that specify the programming of the protocol. The fields within these data structures specify the number of DNIS and ANI digits to expect.

If the line is already configured, the line resets to idle state (any ongoing calls are terminated). An error indication is returned, followed by an indication that the line is correctly initialized.

When using *L4L3.data.cas_data.jate_redial_method*, the following definition applies:

Specifies redial restriction method when country code is JAPAN. Any other value specified in *jate_redial_method* defaults to IISDN_JATE_REDIAL_2IN3_MINS. Redial restrictions include:

IISDN_JATE_REDIAL_3IN3_MINS	0
Redial restriction applies after 3 call attempts until 3-minute timer expires.	
IISDN_JATE_REDIAL_15X	1
The redial restriction applies after 15 call attempts.	
IISDN_JATE_REDIAL_NO_RESTRICT	2
No redial restriction applies.	

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.lapdid
- **L4_to_L3_struct**.data.cas_data.bchannel
- **L4_to_L3_struct**.call_ref set to: ((Trunk <<8) | Bchannel)
- **L4L3**.data.cas_data.signaling_type --- IISDNsigtypeR2_MF

The data structure **_IISDN_R2_INTERREGISTER_PARAMS** specifies these parameters and is defined in *iisdn.h*. See *Table 14* for the structure fields used with the *L4L3mENABLE_CAS* message.

Table 14. _IISDN_R2_INTERREGISTER_PARAMS Fields

Fields	Meaning
dnisMaxNumDigits	Maximum number of DNIS digits to collect
aniMaxNumDigits	Maximum number of ANI digits to collect

Expected Response

The module responds with *L3L4mCAS_STATUS* and **L3_to_L4_struct**.data.b_channel_status set to IISDNbcsIN_SERVICE.

Data Checking

The *L4L3mENABLE_CAS* message checks the value of the E1 trunk and the B-channel whether the B-channel is checked. The B-channel value must be between 0-29.

Error Codes

L3L4errD_CHAN_NOT_DISABLED	The specified B-channel was already running a protocol. The line is reset and configured to run the new protocol.
L3L4errINITIALIZATION_FAILED	Protocol-specific initialization failed.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in *L4_to_L3_struct.data.cas_data.signaling_type* structure.
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks are present on the module.
L3L4errNO_CRSTRUCT_AVAILABLE	No BOSTON channel is available to perform digit detection/generation for the B-channel.
L3L4errPROTOCOL_DISABLED	Protocol specified in structure *L4_to_L3_struct.data.cas_data.signaling_type* is not supported.

See Also

IISDN_BCHANNEL_ID, IISDN_E1_CAS_R2_DATA

Example

```
L4_to_L3_struct L4L3;
memset(&L4L3, 0, sizeof(L4_to_L3_struct)); // zero data structure
L4L3.msgtype = L4L3mENABLE_CAS;
L4L3.data.cas_data.bchannel = 15;          // a B-channel number
L4L3.lapdid  = 0;                          // a Trunk number

L4L3.data.cas_data.signaling_type = IISDNsigtypeR2_MF;

//Fill in the country specific information.  Note: Illegal choice defaults //to
Argentina.  r2_Argentina and r2_Brazil are defined in the include
//files r2Argentina.h and r2Brazil.h respectively.

if (Country == ARGENTINA)
  memcpy((unsigned char*)&L4L3.data.cas_data.cas_params.e1_cas_r2_data,
          (unsigned char*)&r2_Argentina, sizeof(IISDN_E1_CAS_R2_DATA));
else if (Country == BRAZIL)
  memcpy((unsigned char*)&L4L3.data.cas_data.cas_params.e1_cas_r2_data,
          (unsigned char*)&r2_Brazil, sizeof(IISDN_E1_CAS_R2_DATA));
else
  memcpy((unsigned char*)&L4L3.data.cas_data.cas_params.e1_cas_r2_data,
          (unsigned char*)&r2_Argentina, sizeof(IISDN_E1_CAS_R2_DATA));

BsmiControlWrite(fd, &L4L3); // send command to the board
```

L4L3mINFO_REQUEST

Description

Sends a sequence of R2 MF tones. This message is used for test purposes only.

Arguments

The trunk must always be specified in **`L4_to_L3_struct`**`.lapdid`.

The B-channel is determined by setting **`L4_to_L3_struct`**`.call_ref` to: `((Trunk <<8) | Bchannel)`.

Expected Response

None.

Data Checking

The timeslot must be in the range 0 through 29.

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in a connected state.

All R2 MF tones to be dialed are verified. Accepted tokens:

- 0 - 9 (tones 10, 1-9)
- B - F (tones 11-15)

To dial tone 10, specify 0 (not A, invalid).

Error Codes

L3L4errINVALID_COMMAND_ARGS	Invalid data in *L4_to_L3_struct.data* structure.
L3L4errINVALID_MSG_FOR_STATE	Message sent while the protocol was in a state that did not accept it. *L4L3mCALL_PROCEEDING_REQUEST* is only accepted when a call is in a connected (answer) state.
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks are present on the module.

See Also

IISDN_INFO_DATA, *Called Party (IISDN_CALLED_PARTY)*

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mINFO_REQUEST;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.call_ref = ((L4L3m.lapdid << 8) | 20);
L4L3.data.info_data.called_party.num_digits = 5;
strncpy(L4L3.data.info_data.called_party.digits, “205Bc”, 5);
BsmiControlWrite (fd, &L4L3);
```

L4L3mREQ_ABCD_DATA

Description

Requests the module to return all signaling bits (received and transmitted) from all B-channels in module to the host.

Arguments

The trunk must always be specified in **`L4_to_L3_struct`**`.lapdid`.

Expected Response

The module responds with *L3L4mABCD_SIGNAL_DATA*. See description of *IISDN_ABCD_DATA* and *IISDN_ABCD_SIGNALS* for more details.

Data Checking

The trunk number is verified, as well as the trunk type. Only modules with digital trunks (T1 and E1) accept this message.

Error Codes

L3L4errINVALID_INTERFACE	Message not supported for this type of interface (for example, CAS signaling bits cannot be set or retrieved on analog lines).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks are present on the module.

See Also

IISDN_CONFIG_DATA, IISDN_ABCD_DATA, IISDN_ABCD_SIGNALS

Example

```
L4_to_L3_struct L4L3;
memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mREQ_ABCD_DATA;

//This example uses trunk 0
L4L3m.lapdid = 0;
L4L3m.call_ref = 0;

BsmiControlWrite(fd, &L4L3);
```

L4L3mSET_CAS_SIGNALING_BITS

Description

Sets the signaling bit pattern on a particular B-channel.

Arguments

The trunk must always be specified in **`L4_to_L3_struct`**`.lapdid`.

The B-channel is determined by setting **`L4_to_L3_struct`**`.call_ref` to: `((Trunk <<8) | Bchannel)`.

Expected Response

None.

Data Checking

The timeslot must be in the range 0 through 29 for E1 or 0 through 23 for T1.

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

This message can only be used if the port is configured to use transparent signaling (field **`L4_to_L3_struct`**`.data.cas_data.signalling_type` set to `IISDNsigtypeTRANSPARENT`).

If the port is configured to use any other protocol (LEC or R2), this message interferes with the correct behavior of the protocol stack.

Error Codes

L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_INTERFACE	Message is not supported for this type of interface (for example, CAS signaling bits cannot be set or retrieved on analog lines).
L3L4errTOO_MANY_LINKS	The number of entries specified in *L4_to_L3_struct.data.cas_signaling_bits.n_entries* is greater than the maximum allowed (defined by constant IISDN_MAX_CAS_SIGNALING_BIT_ENTRIES).
L3L4mFEATURE_NOT_ACTIVE	Channel configuration does not support this option. An attempt was made to allow the application to set transmitted signal bits when the channel was already configured, or the application did not enable the option to set transmitted signaling bits when CAS_SIGNALING_BIT mode was started.

Example

```
L4_to_L3_struct L4L3;
memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mSET_CAS_SIGNALING_BITS;


BsmiControlWrite(fd, &L4L3);
```

Stack to Application (Module to Host) Messages

R2 Signaling Module to Host message summaries are shown in:

- *Table 15, R2 Signaling L3L4 Management Messages*
- *Table 16, R2 Signaling L3L4 Call Control Messages*

Table 15. R2 Signaling L3L4 Management Messages

Event	Meaning
L3L4mCAS_SIGNALING_BIT_STATUS	Notifies the host that signaling bits for a B-channel have changed.
L3L4mCAS_STATUS ID: 0x40	Enables and disables CAS messages.
L3L4mERROR ID: 0x20	Indicates that a command sent to the module with invalid data.

Table 16. R2 Signaling L3L4 Call Control Messages

Event	Meaning
L3L4mALERTING ID: 0x03	Notifies the outbound side that the call has been accepted by the remote end and the phone is ringing.
L3L4mCAS_CHAN_BLOCKED ID: 0x4B	Responds to a block message or an asynchronous event indicating that the line has been blocked by the remote end.
L3L4mCAS_CHAN_UNBLOCKED ID: 0x4C	Responds to an unblock message or an asynchronous event, indicating that the line was unblocked by the remote end.
L3L4mCLEAR_REQUEST ID: 0x06	Responds to a CLEAR_REQUEST message. This event is not received until the remote end has gone to idle.
L3L4mCONN_ACK_IND ID: 0x0C	Sent as an acknowledge to a *L4L3mCONNECT* message.
L3L4mCONNECT ID: 0x04	Indicates the outbound side when the call is connected.

Table 16. R2 Signaling L3L4 Call Control Messages (Continued)

Event	Meaning
L3L4mDISCONNECT ID: 0x05	Indicates that the remote end has disconnected.
L3L4mPRE_SEIZE ID: 0x33	Indicates that the line was seized and that an incoming call is in progress.
L3L4mSETUP_IND ID: 0x01	Indicates that an incoming call is present. The called and calling party numbers are contained within the L3L4 data structure.

Arguments

Data is passed from the R2 protocol to the host application using the standard ISDN Layer 3 to Layer 4 (**`L3_to_L4_struct`**) data structure.

The protocol passes the call reference number to the application in the *L3L4mSETUP_IND* message for inbound calls and in both the *L3L4mALERTING* and the *L3L4mCONNECT* messages for outbound calls. The field that contains the call reference number is **`L3L4`**`.call_ref`.

Normal Event Sequence

In the call flows given below:

- *R2 -> App* represents events generated by the R2 protocol and sent to the host application.
- *App -> R2* represents messages sent by the host application to the R2 protocol stack.

Inbound Calls

```
R2  -> App:  Trunk:B-channel = 1: 5
             Event = L3L4mPRE_SEIZE

R2  -> App:  Trunk:B-channel = 1: 5
             Event = L3L4mSETUP_IND

App ->  R2:  Trunk:B-channel = 1: 5
             Message = L4L3mCONNECT_REQUEST

R2  -> App:  Trunk:B-channel = 1: 5
             Event = L3L4mCONN_ACK_IND
```

Assume that the outbound side initiates disconnect.

```
R2  -> App:  Trunk:B-channel = 1: 5
             Event  = L3L4mDISCONNECT

App ->  R2:  Trunk:B-channel = 1: 5
             Message = L4L3mCLEAR_REQUEST

R2  -> App:  Trunk:B-channel = 1: 5
             Event  = L3L4mCLEAR_REQUEST
```

Outbound Calls

```
App ->  R2:  Trunk:B-channel = 0: 5
             Message = L4L3mCALL_REQUEST

R2  -> App:  Trunk:B-channel = 0: 5
             Event = L3L4mALERTING

R2  -> App:  Trunk:B-channel = 0: 5
             Event = L3L4mCONNECT
```

Assume that the outbound side initiates disconnect.

```
App ->  R2:  Trunk:B-channel = 0: 5
              Message = L4L3mCLEAR_REQUEST

R2  -> App:  Trunk:B-channel = 0: 5
             Event  = L3L4mCLEAR_REQUEST
```

R2 Signaling L3L4 Messages

The following message subset is used in the R2 signaling protocol for Module to Host messaging. Specific message details begin on:

- *L3L4mALERTING on page 913*
- *L3L4mCAS_CHAN_BLOCKED on page 914*
- *L3L4mCAS_CHAN_UNBLOCKED on page 916*
- *L3L4mCAS_SIGNALING_BIT_STATUS on page 917*
- *L3L4mCAS_STATUS on page 919*
- *L3L4mCLEAR_REQUEST on page 921*
- *L3L4mCONN_ACK_IND on page 923*
- *L3L4mCONNECT on page 924*
- *L3L4mDISCONNECT on page 925*
- *L3L4mERROR on page 927*
- *L3L4mPRE_SEIZE on page 929*
- *L3L4mSETUP_IND on page 930*

L3L4mALERTING

Description

This message notifies the application that:

- An outbound call is successfully initiated
- The called party is notified
- The module is monitoring the line to detect an answer condition

Note that some protocols do not provide an answer indication.

Arguments

Data is specified in:

L3_to_L4_struct.data.al_con_data

See Also

Alerting and Connecting Data Message (IISDN_AL_CON_DATA), *IISDN_ABCD_SIGNALS*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mALERTING)
{
   printf("L3L4mALERTING B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mCAS_CHAN_BLOCKED

Description

Notifies the host that the channel is blocked. The message is sent as a result of either:

- The host blocking the line (*L4L3mCAS_CHAN_BLOCK*)
- Network explicitly blocking the line
- The module blocking the line because of an internal error

L3_to_L4_struct.data.b_channel_status indicates the reason.

If the channel is blocked:

- By the host: the host can unblock it an any time by sending *L4L3mCAS_CHAN_UNBLOCK*
- By the network: Only the network can unblock it - sending *L4L3mCAS_CHAN_UNBLOCK* is not allowed. It issues an error message.
- By an internal error: report occurrence to Dialogic Technical Services and Support. See *Getting Technical Support on page 27*.

The application cannot initiate or receive calls when the line is blocked. Trying to initiate an outbound call results in an error.

Both the network and the host can block a line. When this happens, the line remains unusable until both ends unblock it. Unblock the line at the near end by sending *L4L3mCAS_CHAN_UNBLOCK*. This indicates that the module is ready to initiate and receive calls. However, no calls can actually happen until the network also indicates that it is ready to initiate and receive calls.

The possible values for **L3_to_L4_struct**.data.b_channel_status:

R2_BLOCKED_REMOTE_BLOCKING	Network blocked line.
R2_BLOCKED_NEAR_END_BLOCKING	Host blocked line.
R2_BLOCKED_IDLE_TIMEOUT	Network did not idle within a specified time.

R2_BLOCKED_NO_RESOURCES	Signaling bit monitoring could not start.
R2_BLOCKED_NO_RESPONSE	Error in R2 MF tone detector.

Arguments

Data is specified in: **`L3_to_L4_struct`**`.data.b_channel_status`.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCAS_CHAN_BLOCKED)
{
   printf("BChannel (%02d, %02d)", L3L4.lapdid, L3L4.bchannel);
   switch (l3L4.data.b_channel_status)
   {
   case R2_BLOCKED_REMOTE_BLOCKING:
      printf("Line blocked by network");
      break;
   case R2_BLOCKED_NEAR_END_BLOCKING:
      printf("Line blocked by host");
      break;
   case R2_BLOCKED_IDLE_TIMEOUT;
      printf("Line blocked by network - remained in clearback state");
      break;
   case R2_BLOCKED_NO_RESOURCES;
      printf("Internal error");
      break;
   case R2_BLOCKED_NO_RESPONSE;
      printf("Internal error in R2 MF signaling");
      break;
   default:
      printf("Unknown reason");
      break;
   }
   printf("\n");
}
```

L3L4mCAS_CHAN_UNBLOCKED

Description

Notifies the host that the channel is unblocked. This results when either:

- The host uses *L4L3mCAS_CHAN_UNBLOCK*
- The network unblocks their end of the line

Both the network and the host can block a line. When this happens, the line remains unusable until both ends unblock it. Unblock the line at the near end by sending *L4L3mCAS_CHAN_UNBLOCK*. This indicates that the module is ready to initiate and receive calls. However, no calls can actually happen until the network also indicates that it is ready to initiate and receive calls.

Unblocking the line at the near end by sending *L4L3mCAS_CHAN_UNBLOCK* indicates that the module is ready to initiate and receive calls. However, no calls can actually happen until the network also indicates that it is ready to initiate and receive calls.

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCAS_CHAN_UNBLOCKED)
{
   printf("BChannel (%02d, %02d) Line Unblocked\n", L3L4.lapdid,
   L3L4.bchannel);
}
```

L3L4mCAS_SIGNALING_BIT_STATUS

Description

Notifies the host that signaling bits for a B-channel have changed.

This message is generated by the module every time there is a signaling bit change (either received or transmitted) for any channel.

- The network controls changes in the received bits.
- The module controls changes in the transmitted bits either as a consequence of reception of an L4L3 message or in response to a network signal.

One single *L4L3mCAS_SIGNALING_BIT_STATUS* can carry information about signaling bit changes in several channels.

This message is only generated by a module with digital trunks configured to run CAS protocols.

Arguments

Data is specified in:

`L3_to_L4_struct``.data.cas_signaling_bits`.

See Also

IISDN_CAS_SIGNALING_BITS_DATA, *IISDN_CAS_SIGNALING_BITS*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCAS_SIGNALING_BIT_STATUS)
{
   printf("L3L4mCAS_SIGNALING_BIT_STATUS trunk=%02d\n",
      L3L4.lapdid);
   for (EntryIndex = 0; EntryIndex <
      L3L4.data.cas.signaling_bits.n_entries; EntryIndex++)
   {
      switch (L3L4.data.cas_signaling_bits.entry[EntryIndex].command)
      {
      case CAS_EVENT_RX_CHANGE;
          printf("  Change in received bits B-channel %02d Rx=%02X
            Tx=%02X\n",
            L3L4.data.cas_signaling_bits_entry[EntryIndex].bchannel,
            L3L4.data.cas_signaling_bits_entry[EntryIndex].rx_bits,
            L3L4.data.cas_signaling_bits_entry[EntryIndex].tx_bits);
          break;
      case CAS_EVENT_TX_CHANGE;
          printf("  Change in received bits B-channel %02d Rx=%02X
            Tx=%02X\n",
            L3L4.data.cas_signaling_bits_entry[EntryIndex].bchannel,
            L3L4.data.cas_signaling_bits_entry[EntryIndex].rx_bits,
            L3L4.data.cas_signaling_bits_entry[EntryIndex].tx_bits);
          break;
      }
   }
}
```

L3L4mCAS_STATUS

Description

This message is sent to indicate changes in the initialization state of a line received. For example: as a result of a *L4L3mENABLE_CAS*.

Arguments

Data is specified in:

L3_to_L4_struct.data.b_channel_status.

Values include:

IISDNbcsIN_SERVICE	Indicates the channel is successfully initialized.
IISDNbcsOUT_SERVICE	Indicates the channel is taken out of service.

See Also

Basic type IISDNu8bit

Example

```
L3_to_L4_struct L3L4;
BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCAS_STATUS)
{
   printf("L3L4mCAS_STATUS B-channel %02d:%02d call_ref=%04X,
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);

   switch (L3L4.data.b_channel_status)
   {
      case IISDNbcsIN_SERVICE:
      printf("IN_SERVICE");
      break;

      case IISDNbcsOUT_OF_SERVICE:
      printf("OUT_OF_SERVICE");
      break;
   }
   printf("\n");
}
```

L3L4mCLEAR_REQUEST

Description

This message notifies the host that the line is idle and available for another call (inbound or outbound). This normally requires that both the module and the far end (network) take action to release the call.

If the network disconnects a call first, the module detects and notifies the host through *L3L4mDISCONNECT*. It either releases the line automatically or waits for the host to release the line dependent on the following value set to `TRUE` when the line protocol specified with *L4L3mENABLE_CAS*:

`L4_to_L3_struct``.`
`data.cas-data.cas_params.robbed_bit_data.delayed_onhook`

If the host initiates the disconnect (by sending *L3L4mCLEAR_REQUEST*), the module releases the line and waits until the network releases its end of the line. (Until this happens the line cannot be used for another call.) *L3L4mDISCONNECT* is sent to the host to indicate the network has released the line.

Once both ends of the line are released, the module waits for the period specified by the following, and then sends *L3L4mCLEAR_REQUEST* to the host indicating the line is idle:

`L4_to_L3_struct``.`
`data.cas_data.cas_params.robbed_bit_data.guard_interval_timer`

Arguments

Data is specified in: **`L3_to_L4_struct`**`.data.clr_data`.

See Also

IISDN_CLR_DATA, *Cause Data (IISDN_CAUSE)*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCLEAR_REQUEST)
{
   printf("L3L4mCLEAR_REQUEST B-channel %02d:%02d call_ref=%04X
      cause=%d\n",
   L3L4.lapdid, L3L4.bchannel, L3L4.call_ref,
      L3L4.data.clr_data.cause.cause_val);
}
```

L3L4mCONN_ACK_IND

Description

Indicates to the host that it answered the current incoming call in response to *L4L3mCONNECT_REQUEST* and the call is now in connected (answer) state.

If the network must acknowledge the connection and complete the path between calling and the called party (for example: remove ring voltage from the line), the module waits for before sending *L3L4mCONN_ACK_IND* to the host.

In analog lines, the host can specify whether and for how long the module must wait for the presence of loop current before sending *L3L4mCONN_ACK_IND*. This is controlled by the following, sent with *L4L3mENABLE_CAS*:

L4_to_L3_struct.
data.cas_data.cas_params.robbed_bit_data.ignore_loop_current
L4_to_L3_struct.
data.cas_data.cas_params.robbed_bit_data.loop_current_timer

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCONN_ACK_IND)
{
   printf("L3L4mCONN_ACT_IND B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mCONNECT

Description

This message notifies the host that the ongoing outbound call is answered. Not all protocols can detect an answer condition. If this is the case, it is up to the application to use other methods (for example: using Call Progress Monitoring) to detect when the call is answered.

Arguments

Data is specified in:

L3_to_L4_struct.data.al_con_data.

See Also

Alerting and Connecting Data Message (IISDN_AL_CON_DATA)

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCONNECT)
{
   printf("L3L4mCONNECT B-channel %02d:%02d call_ref=%04X\n",
   L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mDISCONNECT

Description

Indicates to the host that the network released the line, either initiating the call disconnection procedure or in response to a disconnection initiated by the module.

A disconnect signal must be present on the line (for recognition) for a duration of at least:

L4_to_L3_struct.
`data.cas-data.cas_params.robbed_bit_data.hooktimer_onhook_rls`

If the signal remains on the line for a shorter duration, it is interpreted as a hookflash if the protocol supports it. Otherwise it is ignored.

If the network disconnects a call first, the module detects and notifies the host through *L3L4mDISCONNECT*. It either releases the line automatically or waits for the host to release the line, dependent on the following value set to `TRUE` when the line protocol is specified with *L4L3mENABLE_CAS*:

L4_to_L3_struct.
`data.cas-data.cas_params.robbed_bit_data.delayed_onhook`

If the host initiates the disconnect (*L4L3mCLEAR_REQUEST*), the module releases the line and waits until the network has released its end of the line. (Until this happens the line cannot be used for another call.) *L3L4mDISCONNECT* is sent to the host to indicate the network has released the line.

Once both ends of the line are released, the module waits for the period specified by the following, and then sends *L3L4mCLEAR_REQUEST* to the host indicating the line is idle:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.guard_interval_timer`

Arguments

Data is specified in:

L3_to_L4_struct.`data.clr_data`.

See Also

IISDN_CLR_DATA, *Cause Data (IISDN_CAUSE)*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mDISCONNECT)
{
   printf("L3L4mDISCONNECT B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mERROR

Description

Indicates that the last message received from the host contained an error. *Table 17* provides a general explanation for the error message meaning. You can find specific error details in the message that caused the error.

Arguments

Data is specified in: **`L3_to_L4_struct`**`.data.R2_error`.

Table 17. BSMI Error Codes

Error Code	Meaning
L3L4errCALL_REF_ERROR	B-channel values obtained from the data structure associated to a message from *L4_to_L3_struct.call_ref* did not match.
L3L4errD_CHAN_NOT_DISABLED	The specified B-channel was already running a protocol. The firmware resets the line, and configures it to run the new protocol.
L3L4errFEATURE_NOT_ACTIVE	Message not currently supported because the current channel mode was started without supporting this option.
L3L4errGLARE	A signal indicating presence of an inbound call was received when the module was trying to initiate an outbound call.
L3L4errINITIALIZATION_FAILED	Protocol-specific initialization failed.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded maximum number of available B-channels (30 for E1, 24 for T1).
L3L4errINVALID_COMMAND_ARGS	Data structure associated to a message has incorrect values.
L3L4errINVALID_INTERFACE	Message is not supported for this type of interface (for example: analog line coefficients cannot be downloaded to digital (T1/E1) lines).

Table 17. BSMI Error Codes (Continued)

Error Code	Meaning
L3L4errINVALID_MSG_FOR_STATE	Message is not accepted in the current protocol state (for example: send *L4L3mCONNECT_REQUEST* to answer a call when no incoming call is present).
L3L4errINVALID_SMI_MSGID	Message is not recognized by the protocol (if protocol is running, the message was sent to the module in error). For example, any of the following get an error: *L4_to_L3_struct.lapdid* *L4_to_L3_struct.call_ref* *L4_to_L3_struct.*xxx*.bchannel*
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks are present on the module.
L3L4errNO_CRSTRUCT_AVAILABLE	No BOSTON channel is available to perform digit detection/generation for the B-channel.
L3L4errPROTOCOL_DISABLED	Protocol specified is not supported by the module.
L3L4errTOO_MANY_LINKS	The number of specified entries is greater than the maximum supported for the command.

See Also

IISDN_R2_ERROR

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);

//check error code here

if (L3L4.msgtype == L3L4mERROR)
{
   printf("BChannel (%02d,%02d) Error %d (command=%02X State=%04X)\n",
      L3L4.lapdid, L3L4.bchannel,
      L3L4.data.r2_error.error_code,
      L3L4.data.r2_error.CurrentCommand,
      L3L4.data.r2_error.state);
}
```

L3L4mPRE_SEIZE

Description

This message is sent by the protocol to notify the host of the seizure of the line by the network. At this point, the incoming call is not yet fully established or characterized. Following transmission of this message to the host, the protocol acknowledges the incoming call and starts the R2 MFC signaling. Once the R2 MFC signaling ends, the protocol sends *L3L4mSETUP_IND* to the host.

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mPRE_SEIZE)
{
   printf("L3L4mPRE_SEIZE B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mSETUP_IND

Description

Notifies the host of an incoming call, and provides the Called Party address. The host must indicate whether to accept or reject the call.

- Use *L4L3mCONNECT_REQUEST* to accept and answer the call.
- Use *L4L3mALERTING_REQUEST* to accept the call but not answer it (the module sends back a ringback tone to calling party).
- Use *L4L3mCLEAR_REQUEST* to reject the call.

The notification of how to handle the call must be sent to the module in a very short period of time since the module needs this information to proceed with the R2 MF signaling.

Arguments

Data is specified in: **`L3_to_L4_struct`**`.data.setup_data`.

See Also

IISDN_SETUP_DATA, *Calling Party (IISDN_CALLING_PARTY)*, *Called Party (IISDN_CALLED_PARTY)*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mSETUP_IND)
{
   printf("L3L4mSETUP_IND B-channel %02d:%02d call_ref=%04X"
      "call_type=%d calling_party=%s called party=%s\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref,
      L3L4.data.setup_data.call_type,
      L3L4.data.setup_data.calling_party_digits,
      L3L4.data.setup_data.called_party_digits);
}
```

25 - LEC Protocols with BSMI

This chapter explains the control messages provided when using a Local Exchange Carriers (LEC) protocol.

It has the following sections:

- *Application to Stack (Host to Module) Messages*
- *LEC Signaling L4L3 Messages*
- *Stack to Application (Module to Host) Messages*
- *LEC Signaling L3L4 Messages*

The Local Exchange Carriers (LEC) protocols within Local Access and Transport Area (LATA) networks are commonly run over T1 trunks using Robbed-Bit Signaling (RBS).

Most protocols are defined in terms of call states and events, not by line type (E1, T1 or analog). Protocols can also be on various line types. This BSMI implementation does not tie a protocol to a particular line interface, but verifies that the line interface can support the selected signaling type. For example, determining whether Loop Start protocol can run on T1, E1, or Analog loop start lines. The LEC protocols include:

- Wink Start
- Immediate Start
- Delay Dial
- Loop Start (Station and Office)
- Ground Start (Station and Office)

Application to Stack (Host to Module) Messages

For Host to Module message summaries for LEC protocols, see:

- *Table 18, LEC Protocol L4L3 Management Messages*
- *Table 19, LEC Protocol L4L3 Call Control Messages*

Table 18. LEC Protocol L4L3 Management Messages

Message	Meaning
L4L3mDISABLE_CAS ID: 0xC7	Stops the LEC protocol. This message is issued for each logical channel available on the module.
L4L3mENABLE_CAS ID: 0xC6	Starts the LEC protocol. This message is issued for each logical channel available on the module.
L4L3mREQ_ABCD_DATA ID: 0xBC	Requests the received and transmitted signaling bits for each B-channel on the module.
L4L3mREQ_CONFIGURATION ID: 0xEA	Retrieves parameters on a B-channel or line interface.

Table 19. LEC Protocol L4L3 Call Control Messages

Message	Meaning
L4L3mCALL_REQUEST ID: 0x81	Places an outbound call. The host can specify whether the module or the application dials the digits.
L4L3mCLEAR_REQUEST ID: 0x85	Disconnects a connected call or rejects an incoming one.
L4L3mCOLLECT_DIGITS ID: 0xC9	Instructs the module to start detecting digits.
L4L3mCONNECT_REQUEST ID: 0x84	Answers an inbound call.

Table 19. LEC Protocol L4L3 Call Control Messages (Continued)

Message	Meaning
L4L3mDIAL ID: 0xCA	Specifies digits dialed by the module, and the dialing method (DTMF, MF, or Analog Pulse).
L4L3mEND_DIAL ID: 0xBE	Notifies the module the host has finished dialing digits.
L4L3mFORCE_CONNECTION_REQUEST ID: 0xEB	Notifies the module that an answer condition is detected for the current outbound call. Used when the protocol does not already provide that information.
L4L3mSET_CONFIGURATION ID: 0xE9	Sets parameters on a B-channel or line interface.
L4L3mTX_HOOKFLASH ID: 0xC0	Transmits a hookflash if the line is currently offhook. Not all protocols support sending a hookflash.
L4L3mTX_WINK ID: 0xBD	Transmits a wink if the line is currently onhook. Not all protocols support sending a wink.

Numbering Conventions

The LEC protocols use the standard CAS numbering convention. T1 and E1 trunks are numbered starting from 0, and B-channels are logically numbered starting at 0. A logical circuit (or line interface) is identified by a trunk and a B-channel.

Arguments

Arguments are passed to the LEC protocols using the standard ISDN Layer 4 to Layer 3 (*L4_to_L3_struct*) data structure.

Since the LEC protocols are Channel-Associated Signaling (CAS) protocols (as opposed to Common-Channel Signaling - CCS - as ISDN), each call is directly related to a logical channel. As a result, Call Reference numbers lose much of their importance when compared to ISDN. The LEC and R2 protocols require that the Call Reference number for a certain call/channel always be equal to the following for all call-associated message types:

```
call_ref = (trunk << 8) | (bchannel);
```

Specify the B-channel when messages contain *bchannel* in the data structure.

LEC Signaling L4L3 Messages

Use the following message subset for Host to Module messaging with the LEC protocols. Specific message details begin on *page 936*:

- *L4L3mCALL_REQUEST on page 936*
- *L4L3mCLEAR_REQUEST on page 939*
- *L4L3mCOLLECT_DIGITS on page 941*
- *L4L3mCONNECT_REQUEST on page 943*
- *L4L3mDIAL on page 945*
- *L4L3mDISABLE_CAS on page 948*
- *L4L3mENABLE_CAS on page 950*
- *L4L3mEND_DIAL on page 953*
- *L4L3mFORCE_CONNECTION_REQUEST on page 955*
- *L4L3mREQ_ABCD_DATA on page 957*
- *L4L3mREQ_CONFIGURATION on page 958*
- *L4L3mSET_CONFIGURATION on page 960*
- *L4L3mSET_CONFIGURATION on page 960*
- *L4L3mTX_HOOKFLASH on page 962*
- *L4L3mTX_WINK on page 964*

L4L3mCALL_REQUEST

Description

Starts an outgoing call.

The Called Party number and the Calling Party number are supplied in *Called Party (IISDN_CALLED_PARTY)* and *Calling Party (IISDN_CALLING_PARTY)* respectively.

If the application supplies the Called Party number, the module seizes the line, dials the specified number and starts monitoring for answer.

If the application does not specific the Called Party number, the module seizes the line and then waits to control the dialing process. The host can communicate directly with the Tone Generation facility if it chooses. In that case, it must notify that the module it has finished dialing by issuing an *L4L3mEND_DIAL*. Alternatively, the host can notify the module to effect the dialing by issuing *L4L3mDIAL* (no *L4L3mEND_DIAL* is necessary). The module starts monitoring the line for an answer condition once the dialing is complete.

It is possible for the host to specify only part of the digits in *L4L3mCALL_REQUEST*, then follow up with one or more *L4L3mDIAL* messages to complete dialing.

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.`lapdid`
- **L4_to_L3_struct**.`data.call_req_data.bchannel`
- **L4_to_L3_struct**.`call_ref` set to:
  `((Trunk << 8) | Bchannel)`
- **L4_to_L3_struct.**`data.call_req_data` must be set properly.

Expected Response

If a call is properly initiated, sends *L3L4mPROGRESS* to the host.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channel depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have a variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in an idle state.

Digits specified as Called Party number and Calling Party number are checked for validity.

Error Codes

L3L4errGLARE	A signal indicating presence of an inbound call received when the module was trying to initiate an outbound call.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_CALLED_NUMBER	Invalid data in *L4_to_L3_struct.data.call_req_data* structure. *called_party.digits* must contain valid tokens (each dialing mode supports different tokens: ■ IISDNdigtypeDTMF ■ IISDNdigtypeMF ■ IISDNdigtypePULSE
L3L4errINVALID_COMMAND_ARGS	Invalid data in *L4_to_L3_struct.data.call_req_data. call_party_num_digits* must have a positive value.
L3L4errINVALID_MSG_FOR_STATE	Message sent when the protocol state did not accept it. *L4L3mCALL_REQUEST* is only accepted when the line is idle.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

See Also

Called Party (IISDN_CALLED_PARTY), *Calling Party (IISDN_CALLING_PARTY)*, *IISDN_CALL_REQ_DATA*

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mCALL_REQUEST;

//This example uses trunk 0, channel 20
L4L3.lapdid = 0;
L4L3.data.call_req_data.bchannel = 20;
L4L3.call_ref = ((L4L3.lapdid << 8) | L4L3.data.call_req_data.bchannel;

//Called party number.
L4L3.data.call_req_data.called_party.num_digits = 10;
L4L3.data.call_req_data.called_party.num_type = IISDNnumtUNKNOWN;
L4L3.data.call_req_data.called_party.num_plan = IISDNnumpUNKNOWN;
strncpy( &L4L3.data.call_req_data.called_party.digits[0]),
   "1234567890", 10);

//Calling party number (this is not required by the protocol)
L4L3.data.call_req_data.calling_party.num_digits = 10;
L4L3.data.call_req_data.calling_party.num_type = IISDNnumtUNKNOWN;
L4L3.data.call_req_data.calling_party.num_plan = IISDNnumpUNKNOWN;
strncpy( &L4L3.data.call_req_data.calling_party.digits[0]),
   "0987654321", 10);

BsmiControlWrite(fd, &L4L3);
```

L4L3mCLEAR_REQUEST

Description

Clears a call.

The exact behavior of the module once this message is received depends on the protocol used and the line state.

Calls *L4L3mCLEAR_REQUEST* to initiate a call teardown from a connected state in response to a disconnect signal received from the network or to abandon an inbound or outbound call that has not yet been answered.

The module always waits for the line to return to an idle state before reporting to the host that *L4L3mCLEAR_REQUEST* completed. In some protocols this requires action from the other party, so it can take several seconds. After detecting the idle state (before sending *L3L4mCLEAR_REQUEST* to the host) the module waits for a duration specified in:
IISDN_ROBBED_BIT_DATA.guard_interval_timer

Arguments

The trunk must always be specified in **`L4_to_L3_struct`**`.lapdid`.

The B-channel is determined by setting *L4_to_L3_struct.call_ref* to: `((Trunk << 8) | Bchannel)`.

Expected Response

Once the call is terminated and the line is ready for use with another call, *L3L4mCLEAR_REQUEST* is sent to the host.

Data Checking

Verify the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

There must be a call (inbound or outbound) currently on the line.

Error Codes

L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in *L4_to_L3_struct.data*.
L3L4errINVALID_MSG_FOR_STATE	Message sent when the protocol state did not accept it. The firmware only accepts *L4L3mCLEAR_REQUEST* when a call is in progress (inbound or outbound). Inbound calls can only be cleared once *L3L4mSETUP_IND* is sent to the host. Consequently, you cannot terminate an inbound call that is still characterized (from the moment the line is seized and *L3L4mPRE_SEIZE* can be sent to the host to the moment all necessary data is available and the module sends *L3L4mSETUP_IND* to the host).
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

See Also

IISDN_CLR_DATA

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mCLEAR_REQUEST;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.call_ref = ((L4L3m.lapdid << 8) | 20);
BsmiControlWrite (fd, &L4L3);
```

L4L3mCOLLECT_DIGITS

Description

Collects incoming digits in addition to those already received during incoming call setup (specified in *L4L3mENABLE_CAS*):

L4_to_L3_struct.
`cas_data.cas_params.robbed_bit_data.max_incoming_digit_count`

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.`lapdid`
- **L4_to_L3_struct**.`data.dial_data.bchannel`
- **L4_to_L3_struct**.`call_ref` set to:
  `((Trunk << 8) | Bchannel)`
- **L4_to_L3_struct**.`data.dial_data` must be set properly.

Expected Response

Digits returned to the host through *L3L4mINFO_REQUEST*.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in an inbound seized state (an incoming call is detected), but not answered by the host.

Error Codes

L3L4errCALL_REF_ERROR	B-channel values obtained from *L4_to_L3_struct.data.dial_data.bchannel* and from *L4_to_L3_struct.call_ref* did not match.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in *L4_to_L3_struct.data.dial_data*.
L3L4errINVALID_MSG_FOR_STATE	Message sent when the protocol state did not accept it. *L4L3mCOLLECT_DIGITS* is only accepted when the incoming call is present, and the originating end is ready to send the address (called party) digits.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mCOLLECT_DIGITS;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.dial_data.bchannel = 20;
L4L3m.data.dial_data.num_digits = 4;  //wait for 4 digits
L4L3m.call_ref = ((L4L3m.lapdid << 8) |
   L4L3m.data.dial_data.bchannel);
BsmiControlWrite (fd, &L4L3);
```

L4L3mCONNECT_REQUEST

Description

Answers an incoming call.

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.lapdid
- **L4_to_L3_struct**.data.al_con_data.bchannel
- **L4_to_L3_struct**.call_ref set to:
  ((Trunk << 8) | Bchannel)
- **L4_to_L3_struct**.data.al_con_data must be set properly.

Expected Response

After the connection completes, *L3L4mCONN_ACK_IND* is sent to the host.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have variable number of trunks

Unitize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in an inbound seized state (an incoming call is detected, but not answered by the host).

See Also

Alerting and Connecting Data Message (IISDN_AL_CON_DATA)

Error Codes

L3L4errCALL_REF_ERROR	B-channel values from *L4_to_L3_struct.data.al_con_data.bchannel* and *L4_to_L3_struct.call_ref* did not match.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in *L4_to_L3_struct.data.al_con_data*.
L3L4errINVALID_MSG_FOR_STATE	Message sent when the protocol state did not accept it. *L4L3mCONNECT_REQUEST* is only accepted when the inbound call is present and fully characterized (the complete address (called party number) is available). You can call *L4L3mCONNECT_REQUEST* only once for a particular inbound call.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mCONNECT_REQUEST;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.al_con_data.bchannel = 20;
L4L3m.call_ref = ((L4L3m.lapdid << 8) | L4L3m.data.al_con_data.bchannel);

BsmiControlWrite (fd, &L4L3);
```

L4L3mDIAL

Description

Instructs the protocol to dial digits. The dialing method as specified:

- Pulse Dialing
- DTMF
- MF

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.lapdid
- **L4_to_L3_struct**.data.al_con_data.bchannel
- **L4_to_L3_struct**.call_ref set to:
  ((Trunk << 8) | Bchannel)
- **L4_to_L3_struct.**data.dial_data must be set properly.

Expected Response

Once the dialing is completed, *L3L4mEND_DIAL* is sent to the host if **IISDN_DIAL_DATA.**report_dial_completion is set to TRUE.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in an outbound seized state (an outgoing call is started, but not answered by the network) or connected state.

Verifies all tokens to be dialing. If the protocol finds any invalid token, an error message is returned to the host and no digits are dialed.

Error Codes

L3L4errINVALID_CALLED_NUMBER	Invalid data. *L4_to_L3_struct.data.dial_data.* digits must contain valid tokens. Each dialing mode supports different tokens: ■ IISDNdigtypeDTMF ■ IISDNdigtypeMF ■ IISDNdigtypePULSE
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data. *L4_to_L3_struct.data.dial_data.* *num_digits* must have a positive value. *default_digit_type* must contain a valid dialing type: ■ IISDNdigtypeDTMF ■ IISDNdigtypeMF ■ IISDNdigtypePULSE
L3L4errINVALID_MSG_FOR_STATE	Message sent when the protocol state did not accept it. The protocol only accepts *L4L3mDIAL* when the outbound call is present and the protocol is dialing any digits specified by previous calls to *L4L3mDIAL* or *L4L3mCALL_REQUEST*. When the call is in connected state (answered), call *L4L3mDIAL*.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.
L3L4errSERVICE_NOT_OFFERED	The protocol does not support this particular feature. For example, lines configured with FXO_LOOP_START cannot dial digits.

See Also

IISDN_DIAL_DATA

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mDIAL;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.dial_data.bchannel = 20;
L4L3m.call_ref = ((L4L3m.lapdid << 8) | L4L3m.data.dial_data.bchannel);
L4L3m.data.dial_data.num_digits = strlen(DialString);
memcpy(L4L3m.data.dial_data.digits, DialString,
   L4L3m.data.dial_data.num_digits);
L4L3m.data.dial_data.default_digit_type = IISDNdigtypeDTMF;
L4L3m.data.dial_data.report_dial_completion = TRUE;

// only for WINK_START protocol with feature group
// B and D enabled
L4L3m.data.dial_data.wait_for_ack_wink = FALSE;

BsmiControlWrite (fd, &L4L3);
```

L4L3mDISABLE_CAS

Description

Stops the protocol running on the specified line. Any ongoing calls are automatically terminated.

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.lapdid
- **L4_to_L3_struct.**data.cas_data.bchannel
- **L4_to_L3_struct**.call_ref set to:
  ((Trunk << 8) | Bchannel)
- **L4_to_L3_struct.**data.cas_data must be set properly.

Expected Response

The module responds with *L3L4mCAS_STATUS* and **L3_to_L4_ struct.**data.b_channel_status is set to IISDNbcsOUT_OF_SERVICE.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

Error Codes

L3L4errCALL_REF_ERROR	B-channel values obtained from *L4_to_L3_struct.data.cas_data.bchannel* and from *L4_to_L3_struct.call_ref* did not match.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

Example

```
L4_to_L3_struct L4L3;
memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mDISABLE_CAS;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.data.cas_data.bchannel = 20;
L4L3m.call_ref = ((L4L3m.lapdid << 8) | L4L3m.data.cas_data.bchannel);

BsmiControlWrite (fd, &L4L3);
```

L4L3mENABLE_CAS

Description

Configures the line to run the specified protocol.

If the protocol has already configured the line, the line resets to an idle state (any ongoing calls are terminated). An error indication is returned, followed by an indication that the line is correctly initialized.

Arguments

The trunk and B-channel must always be specified in fields:

- **`L4_to_L3_struct`**`.lapdid`
- **`L4_to_L3_struct.`**`data.cas_data.bchannel`
- **`L4_to_L3_struct`**`.call_ref` set to:
  `((Trunk << 8) | Bchannel)`
- **`L4_to_L3_struct.`**`data.cas_data.signaling_type` specifies the protocol family:
  - IISDNsigtypeR2_MF

    For protocols based on ITU's R2 implementation running on E1 trunks.
  - IISDNsigtypeLEC_NETWORK

    For various protocols normally used in the US and Canada (Local Exchange Carriers, or LEC) running on digital (E1 or T1) trunks.
  - IISDNsigtypeANALOG

    For protocols running on analog lines.
- **`L4_to_L3_struct.`**`data.cas_data.cas_params.e1_cas_r2_data` contains all parameters for R2 protocols.
- **`L4_to_L3_struct.`**`data.cas_data.cas_params.robbed_bit_data` contains parameters for LEC protocols (running either digital or analog trunks). Dialogic provides files containing a binary image of these data structures.

Using the application, copy these protocol files into the appropriate data structure and pass it directly to the module using *L4L3mENABLE_CAS*. Do not modify most of the fields from their default values. See *IISDN_ROBBED_BIT_DATA* and *IISDN_E1_CAS_R2_DATA* for the field list.

Expected Response

The module responds with *L3L4mCAS_STATUS* and **L3_to_L4_struct.**data.b_channel_status set to IISDNbcsIN_SERVICE.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have variable number of trunks

The module must have enough DSP resources to match the number of B-channels being initialized.

The trunk interface must support the selected signaling type.

Error Codes

L3L4errCALL_REF_ERROR	B-channel values obtained from *L4_to_L3_struct.data.cas_data.bchannel* and from *L4_to_L3_struct.call_ref* did not match.
L3L4errD_CHAN_NOT_DISABLED	The specified B-channel was already running a protocol. The line is reset, and configured to run the new protocol.
L3L4errINITIALIZATION_FAILED	Protocol-specific initialization failed.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_COMMAND_ARGS	Invalid data in *L4_to_L3_struct.data.cas_data.signaling_type.*
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.
L3L4errNO_CRSTRUCT_AVAILABLE	No BOSTON channel available to perform digit detection/generation for the B-channel.
L3L4errPROTOCOL_DISABLED	Protocol specified in *L4_to_L3_struct.data.cas_data.siganling_type* is not supported.

See Also

IISDN_BCHANNEL_ID, IISDN_ROBBED_BIT_DATA

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mENABLE_CAS;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.data.cas_data.bchannel = 20;
L4L3m.call_ref = ((L4L3m.lapdid << 8) | L4L3m.data.cas_data.bchannel);

L4L3m.data.data.cas_data.signaling_type = IISDNsigtypeLEC_NETWORK;

If (protocol_type == IISDNttWINK_START
{
   PROTOCOL_HEADER_S ProtocolHeader;
   FILE*             fh = fopen("WinkStart.lec", "rb");
// check value of fh
if (fread(&ProtocolHeader, sizeof(ProtocolHeader), 1, fh) == 1
{
   //validate protocol header
   if (fread(L4L3m.data.cas_data.cas_params.robbed_bit_data,
      ProtocolHeader.ProtocolSize, 1 fh) == 1)
   {
      //adjust necessary parameters
      L4L3m.cas_data.cas_params.
          robbed_bit_data.max_incoming_digit_count = IISDN_MAX_DIGITS;
      //send message
      BsmiControlWrite (fd, &L4L3);
   }
   else
   {
      //handle error
   }
  }
    else
  {
   //handle error
  }
}
```

L4L3mEND_DIAL

Description

This message notifies the module that the host has finished dialing digits, and the module can start monitoring the line for an answer signal.

When no called party digits are specified in *L4L3mCALL_REQUEST*, the module assumes the host either requests the digits to be dialed later (using *L4L3mDIAL*) or dials them by directly interacting with the Signal Generation facility. If the latter approach is chosen, the only way for the protocol to know when the host has finished dialing digits is to receive this message from the host.

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.lapdid
- **L4_to_L3_struct.**data.dial_data.bchannel
- **L4_to_L3_struct**.call_ref set to:
  ((Trunk << 8) | Bchannel)

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS*) completed successfully.

The line is successfully seized for outgoing call (in host dialout state) but not digits have been dialed out because *L4L3mCALL_REQUEST* specified no digits.

Error Codes

L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_MSG_FOR_STATE	Message sent when the protocol state did not accept it. The protocol accepts *L4L3mEND_DIAL* only when an outbound call where the host specified no digits to be dialed in *L4L3mCALL_REQUEST* is present (no *Called Party (IISDN_CALLED_PARTY)* specified).
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

Example

```
L4_to_L3_struct L4L3;
memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mEND_DIAL;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.dial_data.bchannel = 20;
L4L3m.call_ref = ((L4L3m.lapdid << 8) | L4L3m.data.dial_data.bchannel);

BsmiControlWrite (fd, &L4L3);
```

L4L3mFORCE_CONNECTION_REQUEST

Description

Notifies the module that an answer condition is detected. Most protocols support some sort of answer notification, except for 5ESS_LOOP_START and 5ESS_GROUND_START. With these exceptions, the application must use Call Progress (CP) or some other mechanism to detect answer and notify the protocol (send *L4L3mFORCE_CONNECTION_REQUEST*) of the change in call state.

Arguments

The trunk and B-channel must always be specified in fields:

- **L4_to_L3_struct**.lapdid
- **L4_to_L3_struct**.data.al_con_data.bchannel
- **L4_to_L3_struct**.call_ref set to:
  ((Trunk << 8) | Bchannel)
- **L4_to_L3_struct**.data.al_con_data must be set properly.

Expected Response

The module responds with *L3L4mCONNECT*.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in an outbound seized state (an outbound call is detected), but not answered by the host.

Error Codes

L3L4errCALL_REF_ERROR	B-channel values obtained from *L4_to_L3_struct.data.cas_data.bchannel* and from *L4_to_L3_struct.call_ref* did not match.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_MSG_FOR_STATE	Message sent when the protocol state did not accept it. *L4L3mFORCE_CONNECTION_REQUEST* is only accepted in outbound calls when the module is monitoring the line to detect an answer condition after dialing all digits.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

See Also

Alerting and Connecting Data Message (IISDN_AL_CON_DATA)

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mFORCE_CONNECTION_REQUEST;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.al_con_data.bchannel = 20;

L4L3m.call_ref = ((L4L3m.lapdid << 8) | L4L3m.data.al_con_data.bchannel);

BsmiControlWrite (fd, &L4L3);
```

L4L3mREQ_ABCD_DATA

Description

Requests the module to return to the host all signaling bits (received and transmitted) from all B-channels in the module.

Arguments

The trunk must always be specified in field **L4_to_L3_struct**.lapdid.

Expected Response

The module responds with *L3L4mABCD_SIGNAL_DATA*. See description of *IISDN_ABCD_DATA* and *IISDN_ABCD_SIGNALS* for more details.

Data Checking

Verifies the trunk number and the trunk type. Only modules with digital trunks (T1 and E1) accept this message.

Error Codes

L3L4errINVALID_INTERFACE	Message not supported for this type of interface (for example, analog line coefficients cannot be downloaded to digital (T1/E1) lines).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

See Also

IISDN_CONFIG_DATA, IISDN_ABCD_DATA, IISDN_ABCD_SIGNALS

Example

```
L4_to_L3_struct L4L3;
memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mREQ_ABCD_DATA;

//This example uses trunk 0
L4L3m.lapdid = 0;
L4L3m.call_ref = 0;

BsmiControlWrite(fd, &L4L3);
```

L4L3mREQ_CONFIGURATION

Description

Requests the value of a line-associated parameter stored by the module. The list of supported parameters is hardware dependent.

Arguments

The trunk and B-channel must always be specified in fields:

- **`L4_to_L3_struct`**`.lapdid`
- **`L4_to_L3_struct.`**`data.config_data.line_index`
- Set **`L4_to_L3_struct.`**`call_ref` to:
  `((Trunk << 8) | Bchannel)`
- **`L4_to_L3_struct.`**`data.config_data` must be set properly.

Expected Response

The module responds with *L3L4mCONFIGURATION_STATUS*.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have a variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The parameter type and value must be valid for the particular hardware platform used.

Error Codes

L3L4errCALL_REF_ERROR	B-channel values obtained from *L4_to_L3_struct.data.cas_data.config_data.line_index* and from *L4_to_L3_struct.call_ref* did not match.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.
L3L4errPARAMETER_NOT_SUPPORTED	Parameter is not supported by the hardware or protocol. For example, audio gain cannot be set in digital (T1/E1) interfaces.

See Also

IISDN_CONFIG_DATA

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));

L4L3.msgtype = L4L3mREQ_CONFIGURATION;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.config_data.line_index = 20;

L4L3m.call_ref =((L4L3m.lapdid << 8) | L4L3m.data.config_data.line_index);

L3L3m.data.config_data.type = IISDNcfgtypeLINE_INPUT_GAIN;

BsmiControlWrite (fd, &L4L3);
```

L4L3mSET_CONFIGURATION

Description

Sets the value of a line-associated parameter. The list of supported parameters and valid ranges is hardware-dependent.

Arguments

The trunk and B-channel must always be specified in fields:

- **`L4_to_L3_struct`**`.lapdid`
- **`L4_to_L3_struct`**`.data.config_data.line_index`
- **`L4_to_L3_struct`**`.call_ref` set to:
  `((Trunk << 8) | Bchannel)`
- **`L4_to_L3_struct`**`.data.config_data` must be set properly.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have a variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The parameter type and value must be valid for the particular hardware platform used.

Error Codes

L3L4errCALL_REF_ERROR	B-channel values obtained from *L4_to_L3_struct.data.cas_data.config_data.line_index* and from *L4_to_L3_struct.call_ref* did not match.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.
L3L4errPARAMETER_OUT_OF_RANGE	Invalid parameter value. The valid range of values depends on the specific parameter.
L3L4errPARAMETER_NOT_SUPPORTED	Parameter not supported by the hardware or protocol. For example, audio gain cannot be set in digital (T1/E1) interfaces.

See Also

IISDN_CONFIG_DATA

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mSET_CONFIGURATION;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.config_data.line_index = 20;

L4L3m.call_ref = ((L4L3m.lapdid << 8) |
   L4L3m.data.config_data.line_index);

L3L3m.data.config_data.type =
   IISDNcfgtypeLINE_INPUT_GAIN;
L4L3m.data.config_data.par.gain = 5;

BsmiControlWrite (fd, &L4L3);
```

L4L3mTX_HOOKFLASH

Description

Notifies the module to send a hookflash signal (disconnect signal followed by a connect signal) on the line. Not all protocols support transmission of hookflash signals.

The duration of the hookflash signal is specified in **L4_to_L3_struct.**`data.signal_duration_data.duration`. If set to 0, then the default duration specified when enabling the protocol is used:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.hookflash_timer`

Arguments

The trunk must always be specified in **L4_to_L3_struct**.`lapdid`.

- To determine B-channel, set **L4_to_L3_struct**.`call_ref` set to: `((Trunk << 8) | Bchannel)`
- **L4_to_L3_struct**.`data.signal_duration_data` must be set properly.

Expected Response

The module sends *L3L4mTX_HOOKFLASH_END* to the host once the hookflash is sent.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have a variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in a connected state and the protocol must support hookflash signals. Protocols supported defined below:

Support for Hookflash	No Support for Hookflash
WINK_START	FX_LOOP_START
DELAY_DIAL	FXO_GROUND_START
IMMEDIATE_START	
FIXED_PAUSE	
5ESS_LOOP_START	
5ESS_GROUND_START	

Error Codes

L3L4errINVALID_B_CHANNEL	B-channel value exceeded maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_MSG_FOR_STATE	Message sent when the protocol state did not accept it. *L4L3mTX_HOOKFLASH* is only accepted when the call is connected.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

See Also

IISDN_SIGNAL_DURATION_DATA

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mTX_HOOKFLASH;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.config_data.line_index = 20;

L4L3m.call_ref = ((L4L3m.lapdid << 8) | 20;

L3L3m.data.signal_duration_data.duration = 30
   //300ms signal

BsmiControlWrite (fd, &L4L3);
```

L4L3mTX_WINK

Description

Notifies the module to send a wink signal (connect signal followed by a disconnect signal) on the line. Not all protocols support transmission of wink signals.

L4_to_L3_struct.`data.signal_duration_data.duration` specifies the duration of the wink signal. If set to 0, then the default duration specified when the enabling protocol is used:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.dptimer_wink`

Arguments

The trunk must always be specified in **L4_to_L3_struct**.`lapdid`.

- To determine B-channel set **L4_to_L3_struct**.`call_ref` set to: `((Trunk << 8) | Bchannel)`
- **L4_to_L3_struct**.`data.signal_duration_data` must be set properly.

Expected Response

The module sends *L3L4mTXWINK_END* to the host once the wink is sent.

Data Checking

Verifies the trunk and B-channel numbers. The range of valid B-channels depends on the hardware being utilized:

- T1/ISDN trunks have 23 B-channels
- E1/ISDN trunks have 30 B-channels
- Analog Loop Start modules have a variable number of trunks

Initialize the protocol for that line (*L4L3mENABLE_CAS* completed successfully).

The line must be in an inbound seized state (an incoming call is detected, but not answered by the host) and the protocol must support wink signals. How protocols support transmission of wink signals is indicated on the following page.

Support for Wink	No Support for Wink
WINK_START	FX_LOOP_START
DELAY_DIAL	FXO_GROUND_START
	IMMEDIATE_START
	FIXED_PAUSE
	5ESS_LOOP_START
	5ESS_GROUND_START

Error Codes

L3L4errCOMMAND_DISABLED	Protocol does not support winking.
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_MSG_FOR_STATE	Message sent when the protocol state did not accept it. *L4L3mTX_WINK* is only accepted when an incoming call is reported to the application but not yet answered.
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.

See Also

IISDN_SIGNAL_DURATION_DATA

Example

```
L4_to_L3_struct L4L3;

memset(&L4L3, 0, sizeof(L4_to_L3_struct));
L4L3.msgtype = L4L3mTX_WINK;

//This example uses trunk 0, channel 20
L4L3m.lapdid = 0;
L4L3m.data.config_data.line_index = 20;

L4L3m.call_ref = ((L4L3m.lapdid << 8) | 20;

L3L3m.data.signal_duration_data.duration = 20
    //200ms wink

BsmiControlWrite (fd, &L4L3);
```

Stack to Application (Module to Host) Messages

For Module to Host message summaries for LEC protocols, see:

- *Table 20, LEC Protocol L3L4 Management Messages*
- *Table 21, LEC Protocol L3L4 Call Control Messages*, on *page 967*

Table 20. LEC Protocol L3L4 Management Messages

Message	Meaning
L3L4mABCD_SIGNAL_DATA ID: 0x35	Returns both the received and transmitted signaling bits for each B-channel on the module.
L3L4mACK_DOWNLOAD ID: 0x44	Indicates downloading of trunk-specific coefficients is successfully completed (currently applicable to analog modules only).
L3L4mACK_UPLOAD ID: 0x47	Returns the trunk-specific coefficients requested through *L4L3mUPLOAD*. Currently applicable to analog modules only.
L3L4mCAS_SIGNALING_BIT_STATUS ID: 0x4F	Notifies host that signaling bits for a B-channel have changed.
L3L4mCAS_STATUS ID: 0x40	Indicates the status of a line in regards to its readiness for use.
L3L4mCONFIGURATION_STATUS ID: 0x55	Returns to the host the value of a requested line parameter.
L3L4mERROR ID: 0x20	Indicates an error condition.

Table 21. LEC Protocol L3L4 Call Control Messages

Message	Meaning
L3L4mALERTING ID: 0x03	Indicates the outbound call is accepted and the called party is notified (telephone ringing).
L3L4mCALLER_ID_DETECTED ID: 0x59	Notifies host that called ID information is detected on the line.
L3L4mCLEAR_REQUEST ID: 0x66	Notifies the host that line is idle and ready for another call (inbound or outbound).
L3L4mCONN_ACK_IND ID: 0x0C	Notifies the host that the ongoing outbound call is answered.
L3L4mCONNECT ID: 0x04	Notifies host that the request to answer an incoming call is completed successfully.
L3L4mDISCONNECT ID: 0x05	Indicates the far end has disconnected the existing call.
L3L4mEND_DIAL ID: 0x42	Notifies host that the dial request (*L4L3mDIAL*) is successfully completed.
L3L4mHOOKFLASH ID: 0x3E	Notifies host that a hookflash signal is detected on the line.
L3L4mLOOP_ON ID: 0x56	Notifies the host that loop current is detected on the line.
L3L4mLOOP_REVERSAL ID: 0x57	Notifies host that the loop current reversal is detected on the line.
L3L4mPRE_SEIZE ID: 0x33	Notifies host that the line is seized for an incoming call, and the handshaking necessary to proceed with the call is taking place.
L3L4mPROGRESS ID: 0x02	Notifies host that an outbound call is initiated, and the handshaking necessary to proceed with the call is taking place.
L3L4mRING_STATUS ID: 0x58	Notifies host about a change in the status of the ring signal.

Table 21. LEC Protocol L3L4 Call Control Messages (Continued)

Message	Meaning
L3L4mRX_WINK ID: 0x36	Notifies host that a wink is detected on the line.
L3L4mSEIZE_COMP ID: 0x34	Notifies the host that the process for initiating an outbound call successfully completed.
L3L4mSETUP_IND ID: 0x01	Indicates an incoming call is present.
L3L4mSTATUS_IND ID: 0x07	Not currently used.
L3L4mTX_HOOKFLASH_END ID: 0x38	Notifies host that a hookflash is successfully transmitted.
L3L4mTXWINK_END ID: 0x37	Notifies host that a wink is successfully transmitted.

Arguments

Data is passed from the LEC protocol to the host application using the standard ISDN Layer 3 to Layer 4 (*L3_to_L4_struct*) data structure.

The following fields are common to all LEC protocol messages:

lapdid

Contains the trunk number (stream number).

bchannel

Contains the timeslot number. If not applicable to a particular message, it is set to zero.

call_ref

Contains trunk/timeslot information, in the format:

```
((lapdid << 8) | (bchannel))
```

LEC Signaling L3L4 Messages

Use the following message subset for Module to Host messaging with LEC protocols. Specific message details begin on *page 970*:

- *L3L4mABCD_SIGNAL_DATA on page 970*
- *L3L4mACK_DOWNLOAD on page 971*
- *L3L4mACK_UPLOAD on page 972*
- *L3L4mALERTING on page 973*
- *L3L4mCALLER_ID_DETECTED on page 974*
- *L3L4mCAS_SIGNALING_BIT_STATUS on page 977*
- *L3L4mCAS_STATUS on page 979*
- *L3L4mCLEAR_REQUEST on page 980*
- *L3L4mCONFIGURATION_STATUS on page 982*
- *L3L4mCONNECT on page 984*
- *L3L4mCONN_ACK_IND on page 983*
- *L3L4mDISCONNECT on page 985*
- *L3L4mEND_DIAL on page 987*
- *L3L4mERROR on page 988*
- *L3L4mHOOKFLASH on page 991*
- *L3L4mLOOP_ON on page 992*
- *L3L4mLOOP_REVERSAL on page 994*
- *L3L4mPRE_SEIZE on page 995*
- *L3L4mPROGRESS on page 996*
- *L3L4mRING_STATUS on page 997*
- *L3L4mRX_WINK on page 999*
- *L3L4mSEIZE_COMP on page 1000*
- *L3L4mSETUP_IND on page 1001*
- *L3L4mSTATUS_IND on page 1002*
- *L3L4mTX_HOOKFLASH_END on page 1003*
- *L3L4mTXWINK_END on page 1004*

L3L4mABCD_SIGNAL_DATA

Description

Responds to *L4L3mREQ_ABCD_DATA* and returns the value of the transmitted and received signaling bits in all CAS spans of the system.

Arguments

Data is specified in:

L3_to_L4_struct.data.abcd_data

See Also

IISDN_ABCD_DATA, IISDN_ABCD_SIGNALS

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mABCD_SIGNAL_DATA)
{
   int trunk, bchannel;
   printf("L3L4mABCD_SIGNAL_DATA B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
   for trunk = 0; trunk < MAX_IISDN_LINES; trunk++)
   {
      for (bchannel = 0; bchannel <30; bchannel++)
      {
          printf("B-channel %02d:%02d Rx=%02X Tx=%02X\n",
          L3L4.lapdid, L3L4.bchannel,
          GET_RX_BITS(L3L4.data.abcd_data.line[trunk].abcd[bchannel]),
          GET_TX_BITS(L3L4.data.abcd_data.line[trunk].abcd[bchannel]),
      }
   }
}
```

L3L4mACK_DOWNLOAD

Description

Indicate downloading of module coefficients (*L4L3mDOWNLOAD*) is completed successfully. Currently the only download accepted by the protocol stack is that of line coefficients for analog modules, and it must be made on a per-trunk basis.

Arguments

Data is specified in:

L3_to_L4_struct.data.ack_download_data

See Also

IISDN_DOWNLOAD_DATA

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mACK_DOWNLOAD)
{
   printf("L3L4mACK_DOWNLOAD Trunk= %2d\n", L3L4.lapdid);
}
```

L3L4mACK_UPLOAD

Description

Returns to the host the module coefficients requested through a call to *L4L3mUPLOAD*. Currently the only uploads accepted by the protocol stack are line coefficients for analog modules. The call must be made on a per-trunk basis.

Arguments

Data is specified in:

L3_to_L4_struct.data.ack_upload_data

See Also

IISDN_UPLOAD_DATA

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mACK_UPLOAD)
{
   int i;

   printf("L3L4mACK_UPLOAD Trunk= %2d\n", L3L4.lapdid);

   printf("module_id=%d total length=%d\n",
      L3L4.data.ack_download_data.module_id,
      L3L4.data.ack_download_data.length,
      L3L4.data.ack_download_ata.total_length);

      For (I=0; i<L3L4.data.ack_download_data.length; i++)
      {
          printf("%02X", L3L4.data.ack_download_data.buffer[i]);
      }
   printf("\n");}
}
```

L3L4mALERTING

Description

Notifies the application that an outbound call is successfully initiated, the called party is notified, and the module is monitoring the line to detect an answer condition.

Some protocols do not provide an answer indication.

Arguments

Data is specified in:

L3_to_L4_struct.data.al_con_data

See Also

Alerting and Connecting Data Message (IISDN_AL_CON_DATA), *IISDN_ABCD_SIGNALS*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mALERTING)
{
   printf("L3L4mALERTING B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mCALLER_ID_DETECTED

Description

Notifies the application that call setup information (*callID*) is detected (optional message).

The characteristics of the call setup information are defined through the fields in:

`L4_to_l3_struct``.`
`data.cas_data.cas_params.robbed_bit_data:`

send_caller_id_event	Controls transmission
call_setup_message_mode	Sends call setup information: DFSK, DTMF, MF
caller_id_standard	Standards defining the call setup information
caller_id_alerting_signal	Alerts sent by the network prior to transmission of call setup information

The protocol normally sends the *CallerID* between the first and second rings or before the first ring. The application must allow enough time for the network to send the information before answering the call.

- Wait a certain time (country-specific) after receiving the indication of incoming call (*L3L4mSETUP_IND*) before answering.
- Delay the transmission of *L3L4mSETUP_IND* by the module until after *CallerID* information is available.

IISDNcfgtypeSETUP_IND_RING_COUNT (see *L4L3mSET_CONFIGURATION*) allows the application to determine the number of ring cycles before reporting *L3L4mSETUP_IND*. Setting this value to 2 gives the network enough time to send *CallerID*.

The protocol also sends the most significant parts of the *CallID* information (if available) within *L3L4mSETUP_IND*. The complete information, however, is only available through *L3L4mCALLER_ID_DETECTED*.

CallerID information is normally available in protocols used between the network and user terminals; 5ESS_LOOP_START and 5ESS_GROUND_START, as opposed to between switches.

Arguments

Data is specified in:

L3_to_L4_struct.data.call_id_data

See Also

IISDN_CALLER_ID_DATA

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCALLER_ID_DETECTED)
{
   IISDN_CALLER_ID_DATA* pCalleridData;

   printf("L3L4mCALLER_ID_DETECTED B-channel %02d:
      %02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);

   pCalleridData = &(L3L4.data.caller_id_data);

   printf(" Message Type                 0x%02X\n",
         pCallerIdData->MessageType);
   printf(" Message Length               %02X\n",
         pCallerIdData->MessageLength);
   printf("  Date                   %.2s/%.2s\n",
      &(pCalleridData->CidRawData[pCallerIdData->CidDataOffset]),
      &(pCalleridData->CidRawData[pCallerIdData->CidDataOffset + 2]));
   printf("  Time                   %.2s/%.2s\n",
      &(pCalleridData->CidRawData[pCallerIdData->CidDataOffset]),
      &(pCalleridData->CidRawData[pCallerIdData->CidDataOffset + 2]));
   printf("  Calling Number           %.*s\n",
      pCallerIdData->CidNameSize,,
      &(pCallerIdData->CidRawData[pCallerIdData->CidDataOffset]));
   printf("  Name                  %.*s\n",
      pCallerIdData->CidNameSize,,
      &(pCallerIdData->CidRawData[pCallerIdData->CidDataOffset]));
   printf("  RawData:");
   For (i=0; i<pCallerIdData->MessageLength + 2; i++)
   {
      printf("%02X", pCallerIdDat6a->CidRawData[i]);
   }
   printf("\n");
}
```

L3L4mCAS_SIGNALING_BIT_STATUS

Description

The module generates this message every time there is a signaling bit change (either received or transmitted) for any channel.

- Changes in the received bits are controlled by the network.
- Changes in the transmitted bits are controlled by the module and are initiated either as a consequence of reception of an L4L3 message or in response to a network signal.

One single *L4L3mCAS_SIGNALING_BIT_STATUS* can carry information about signaling bit changes in several channels.

The module generates this message only with digital trunks configured to run CAS protocols.

Arguments

Data is specified in:

L3_to_L4_struct.data.cas_signaling_bits

See Also

IISDN_CAS_SIGNALING_BITS_DATA, *IISDN_CAS_SIGNALING_BITS*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCAS_SIGNALING_BIT_STATUS)
{
   printf("L3L4mCAS_SIGNALING_BIT_STATUS trunk=%02d\n",
      L3L4.lapdid);
   for (EntryIndex = 0; EntryIndex <
      L3L4.data.cas.signaling_bits.n_entries; EntryIndex++)
   {
      switch (L3L4.data.cas_signaling_bits.entry[EntryIndex].command)
      {
      case CAS_EVENT_RX_CHANGE;
          printf("  Change in received bits B-channel %02d Rx=%02X,
            Tx=%02X\n",
            L3L4.data.cas_signaling_bits_entry[EntryIndex].bchannel,
            L3L4.data.cas_signaling_bits_entry[EntryIndex].rx_bits,
            L3L4.data.cas_signaling_bits_entry[EntryIndex].tx_bits);
          break;
      case CAS_EVENT_TX_CHANGE;
          printf("  Change in received bits B-channel %02d Rx=%02X,
            Tx=%02X\n",
            L3L4.data.cas_signaling_bits_entry[EntryIndex].bchannel,
            L3L4.data.cas_signaling_bits_entry[EntryIndex].rx_bits,
            L3L4.data.cas_signaling_bits_entry[EntryIndex].tx_bits);
          break;
      }
   }
}
```

L3L4mCAS_STATUS

Description

Indicates changes received in the initialization state of a line. For example: as a result of a *L4L3mENABLE_CAS*.

Arguments

Data is specified in:

L3_to_L4_struct.data.b_channel_status

Values include:

IISDNbcsIN_SERVICE	Indicates the channel is successfully initialized.
IISDNbcsOUT_SERVICE	Indicates the channel is taken out of service.

See Also

Basic type IISDNu8bit

Example

```
L3_to_L4_struct L3L4;
BsmiControlRead (fd, &L3L4);
//check error code here
if (L3L4.msgtype == L3L4mCAS_STATUS)
{
   printf("L3L4mCAS_STATUS B-channel %02d:%02d call_ref=%04X,
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);

   switch (L3L4.data.b_channel_status)
   {
      case IISDNbcsIN_SERVICE:
          printf("IN_SERVICE");
          break;

      case IISDNbcsOUT_OF_SERVICE:
          printf("OUT_OF_SERVICE");
          break;
   }
   printf("\n");
}
```

L3L4mCLEAR_REQUEST

Description

Notifies the host that the line is idle and available for another call (inbound or outbound). This notification normally requires that both the module and the far end (network) take action to release the call.

If the network disconnects a call first, the module detects and notifies the host through *L3L4mDISCONNECT*. It either releases the line automatically or waits for the host to release the line. It is dependent on the following value set to `TRUE` when the line protocol specified with *L4L3mENABLE_CAS*:

`L4_to_L3_struct``.`
`data.cas-data.cas_params.robbed_bit_data.delayed_onhook`

If the host initiates the disconnect (by sending *L4L3mCLEAR_REQUEST*), the module releases the line and waits until the network releases its end of the line. (Until this happens, the line cannot be used for another call.) *L3L4mDISCONNECT* is sent to the host to indicate the network has release the line.

Once both ends of the line are released, the module waits for the period specified by the following, and then sends *L3L4mCLEAR_REQUEST* to the host indicating the line is idle:

`L4_to_L3_struct``.`
`data.cas_data.cas_params.robbed_bit_data.guard_interval_timer`

Arguments

Data is specified in:

`L3_to_L4_struct``.data.clr_data`

See Also

IISDN_CLR_DATA, *Cause Data (IISDN_CAUSE)*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCLEAR_REQUEST)
{
   printf("L3L4mCLEAR_REQUEST B-channel %02d:%02d call_ref=%04X,
      cause=%d\n",
   L3L4.lapdid, L3L4.bchannel, L3L4.call_ref,
      L3L4.data.clr_data.cause.cause_val);
}
```

L3L4mCONFIGURATION_STATUS

Description

Returns to the host the value of a channel parameter requested through a call to *L4L3mREQ_CONFIGURATION*.

The protocol returns the value in the appropriate field of union **IISDN_CONFIG_DATA.**par. The application must verify the type of parameter (specified in **IISDN_CONFIG_DATA.**type) and then access the correct field. For the complete list of parameter types and their associated fields, see the description of *IISDN_CONFIG_DATA*.

Arguments

Data is specified in:

L3_to_L4_struct.data.config_data

See Also

IISDN_CONFIG_DATA

Example

```
L3_to_L4_struct L3L4;
BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCONFIGURATION_STATUS)
{
   printf("L3L4mCONFIGURATION_STATUS B-channel %02d:%02d
      call_ref=%04X\n",L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
   switch(L3L4.data.config_data.type);
   {
      case IISDNcfgtypeLINE_INPUT_GAIN;
          printf("LINE INPUT GAIN=%02d\n",
             L3L4.data.config_data.par.gain);
          break;
      case IISDNcfgtypeSETUP_IND_RING_COUNT;
          printf("NUMBER OF RINGS BEFORE ANSWER=%02d\n",
             L3L4.data.config_data.par.setup_ind_ring_count);
          break;
      // other parameter types here
   }
}
```

L3L4mCONN_ACK_IND

Description

Notifies the host that it answered the current incoming call in response to *L4L3mCONNECT_REQUEST* and the call is now in connected (answer) state.

If the network must acknowledge the connection and complete the path between calling and the called party (for example: remove ring voltage from the line), the module waits before sending *L3L4mCONN_ACK_IND* to the host.

In analog lines, the host can specify whether and for how long the module must wait for the presence of loop current before sending *L3L4mCONN_ACK_IND*. This is controlled by the following, sent with *L4L3mENABLE_CAS*:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.ignore_loop_current`
L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.loop_current_timer`

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCONN_ACK_IND)
{
   printf("L3L4mCONN_ACT_IND B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mCONNECT

Description

Notifies the host that the ongoing outbound call is answered. Not all protocols are capable of detecting an answer condition. If this is the case, it is up to the application to use other methods (for example: using Call Progress Monitoring) to detect when the call is answered.

Arguments

Data is specified in:

L3_to_L4_struct.data.al_con_data

See Also

Alerting and Connecting Data Message (IISDN_AL_CON_DATA)

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mCONNECT)
{
   printf("L3L4mCONNECT B-channel %02d:%02d call_ref=%04X\n",
   L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mDISCONNECT

Description

Notifies the host that the network released the line, either initiating the call disconnection procedure or in response to a disconnection initiated by the module.

A disconnect signal must be present on the line (for recognition) for a duration of at least:

L4_to_L3_struct.
`data.cas-data.cas_params.robbed_bit_data.hooktimer_onhook_rls`

If the signal remains on the line for a shorter duration, it is interpreted as a hookflash if the protocol supports it. Otherwise it is ignored.

If the network disconnects a call first, the module detects and notifies the host through *L3L4mDISCONNECT*. It either releases the line automatically or waits for the host to release the line. It is dependent on the following value set to `TRUE` when the line protocol specified with *L4L3mENABLE_CAS*:

L4_to_L3_struct.
`data.cas-data.cas_params.robbed_bit_data.delayed_onhook`

If the host initiates the disconnect (*L4L3mCLEAR_REQUEST*), the module releases the line and waits until the network releases its end of the line. (Until this happens the line cannot be used for another call.) *L3L4mDISCONNECT* is sent to the host to indicate the network has released the line.

Once both ends of the line are released, the module waits for the period specified by the following, and then sends *L3L4mCLEAR_REQUEST* to the host indicating the line is idle:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.guard_inteval_timer`

Arguments

Data is specified in:

L3_to_L4_struct.`data.clr_data`

See Also

IISDN_CLR_DATA, *Cause Data (IISDN_CAUSE)*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mDISCONNECT)
{
   printf("L3L4mDISCONNECT B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mEND_DIAL

Description

Notifies the host that the module finished dialing all digits specified by *L4L3mDIAL* (optional message). The host can use this information to proceed with the flow of the application (for example: waiting for a notification of call answered, performing call progress monitoring, or dialing more digits).

Transmission of *L4L3mEND_DIAL* is controlled by **L4_to_L3_struct.**data.dial_data.report_dial_completion, sent as part of *L4L3mDIAL*.

L3L4mSEIZE_COMP indicates that the digit dialing specified in *L4L3mCALL_REQUEST* is complete.

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mEND_DIAL)
{
   printf("L3L4mEND_DIAL B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mERROR

Description

Notifies the host of an error condition detected by the protocol stack.

There are several possible causes for errors (see *Table 22*):

- Invalid L4L3m messages (not recognized by the protocol)
- Non-decodable L4L3m messages (containing incorrect/inconsistent data fields)
- L4L3m messages not allowed in a particular protocol state
- Hardware events unexpected at the current protocol state
- Internal protocol errors

Arguments

Data is specified in:

`L3_to_L4_struct.``data.error_code`

Error Codes

Table 22. BSMI Error Codes

Error Code	Meaning
L3L4errCALL_REF_ERROR	B-channel values obtained from the data structure associated to a message from *L4_to_L3_struct.call_ref* did not match.
L3L4errD_CHAN_NOT_DISABLED	The specified B-channel was already running a protocol. The line resets, and is configured to run the new protocol.
L3L4errFEATURE_NOT_ACTIVE	Message is not currently supported because the current channel mode was started without supporting this option.
L3L4errGLARE	A signal indicating presence of an inbound call was received when the module was trying to initiate an outbound call.
L3L4errINITIALIZATION_FAILED	Protocol-specific initialization failed.

Table 22. BSMI Error Codes (Continued)

Error Code	Meaning
L3L4errINVALID_B_CHANNEL	B-channel value exceeded the maximum number of available B-channels (30 for E1, 23 for T1).
L3L4errINVALID_COMMAND_ARGS	Data structure associated to a message has incorrect values.
L3L4errINVALID_INTERFACE	Message is not supported for this type of interface (for example: analog line coefficients cannot be downloaded to digital (T1/E1) lines).
L3L4errINVALID_MSG_FOR_STATE	Message is not accepted in current protocol state (for example: send *L4L3mCONNECT_REQUEST* to answer a call when no incoming call is present).
L3L4errINVALID_SMI_MSGID	Message is not recognized by the protocol (if protocol is running, the message sent to the module in error). For example, any of the following set incorrectly: ■ *L4_to_L3_struct.lapdid* ■ *L4_to_L3_struct.call_ref* ■ *L4_to_L3_struct*.xxx.*bchannel*
L3L4errLAPDID_NOT_ESTABLISHED	The application did not configure the line to run any protocol (by sending *L4L3mENABLE_CAS*).
L3L4errLAPDID_OUT_OF_RANGE	Trunk (span) number is greater than the maximum number of trunks present on the module.
L3L4errNO_CRSTRUCT_AVAILABLE	No BOSTON channel is available to perform digit detection/generation for the B-channel.
L3L4errPROTOCOL_DISABLED	Protocol specified is not supported by the module.
L3L4errTOO_MANY_LINKS	The number of specified entries is greater than the maximum supported for the command.

See Also

Basic data type: *IISDNu8bit*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mERROR)
{
   printf("L3L4mERROR B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mHOOKFLASH

Description

Notifies the host that a hookflash signal is detected on the line (optional message). A hookflash is characterized as a disconnect signal received when the call is in connected state with duration less than:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.hooktimer_onhook_rls`

If the duration exceeds this threshold, it is interpreted as a disconnection, and the protocol sends *L3L4mDISCONNECT*.

Transmission of *L4L3mHOOKFLASH* is controlled by (part of *L4L3mENABLE_CAS*):

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.send_hookflash_event`

Before setting the maximum duration for hookflash duration, check with your carrier to verify whether hookflash signals are supported on the particular installation. In general, if the installation does not support hookflash, it responds to a shorter duration disconnect signal (100-200ms) before acted upon. If hookflash signals are allowed, the threshold is usually 800 ms to 1000 ms.

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mHOOKFLASH)
{
   printf("L3L4mHOOKFLASH B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mLOOP_ON

Description

Notifies the host about the presence of loop current (optional message). The host can configure the protocol to send information about the presence of loop current. When going offhook, configure the module to wait until loop current is present for a certain duration before proceeding. This wait ensures that the line interface is properly connected and capable of carrying audio signals.

All loop current-related parameters are set when the protocol is loaded. For example:

L4L3mENABLE_CAS,
L4_to_L3_struct.data.cas_data.cas_params.robbed_bit_data

send_loop_on_event	Controls the transmission of *L3L4mLOOP_ON*
ignore_loop_current	Determines whether the module waits for loop current before proceeding with the call
hooktimer_offhook_inseize	Specifies loop current must be present for the duration defined when initiating a call as a result of *L4L3mCALL_REQUEST*.
hooktimer_offhook_answer	Specifies loop current must be present for the duration defined when answering the call as a result of *L4L3mCONNECT_REQUEST*.

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mLOOP_ON)
{
   printf("L3L4mLOOP_ON B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mLOOP_REVERSAL

Description

Notifies the host that a loop current reversal is detected on the line (optional message).

Sent as part of *L4L3mENABLE_CAS*, transmission of *L4L3mLOOP_REVERSAL* is controlled by:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.send_loop_reversal_event`

Deglitching of loop current reversal, sent as part of *L4L3mENABLE_CAS*, is controlled by:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.hooktimer_offhook_inseize`

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mLOOP_REVERSAL)
{
   printf("L3L4mLOOP_REVERSAL B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mPRE_SEIZE

Description

Notifies the host (sent by the protocol) that the network seized the line (optional message). At this point, the incoming call is not yet fully established nor characterized. Following transmission of this message to the host, the protocol sends any necessary acknowledgement signals to the network, then starts monitoring the line for call setup information (DID digits, CallerID and others). Once all the steps necessary to identify the incoming call are taken, the protocol sends *L4L3mSETUP_IND* to the host.

As part of *L4L3mENABLE_CAS*, transmission of *L4L3mPRE_SEIZE* is controlled by:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.send_preseize_event`

If this field is set to `FALSE`, the host is not notified of the incoming call until all the call setup information is available, although the line is considered to be in a non-idle state, therefore preventing the host from initiating any outbound calls. Regardless of whether *L4L3mPRE_SEIZE* is sent, the host cannot reject the message (by sending *L4L3mCLEAR_REQUEST*) until *L4L3mSETUP_IND* is received.

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mPRE_SEIZE)
{
   printf("L3L4mPRE_SEIZE B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mPROGRESS

Description

Notifies the application that the procedure for establishing an outbound call (initiated by *L4L3mCALL_REQUEST*) is in progress. This message is sent after the module has successfully seized the line, but before any necessary acknowledgement from the network is received, and before any digits are dialed. The completion of the call setup procedure is signaled by transmission of *L3L4mSEIZE_COMP*.

Currently, *L3L4mPROGRESS* returns no data, and all fields in *progress_data* are set to 0.

Arguments

Data is specified in:

L3_to_L4_struct.data.progress_data

See Also

Progress Indication (IISDN_PROGRESS)

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mPROGRESS)
{
   printf("L3L4mPROGRESS B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mRING_STATUS

Description

Notifies the host about the ring cadence (optional message).

The host can configure the protocol to send information about the raw or deglitched ring cadence.

- For the raw cadence, the module notifies the host every time there is a change in the status of the ring voltage detected on the line.
- For deglitched cadence, the module waits until ring voltage is present for the duration defined by:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.hooktimer_min_ring_on`

Or is absent for the duration defined by:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.hooktimer_min_ring_off`

Protocols that use ring voltage (or its digital equivalent) to signal an incoming call generate this message, for example:

- 5ESS_LOOP_START
- 5ESS_GROUND_START

Arguments

Data is specified in:

L3_to_L4_struct.`data.ring_status_data`

See Also

IISDN_RING_STATUS_DATA

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mRING_STATUS)
{
   printf("L3L4mRING_STATUS B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
   printf("  %s RING %s; %d cycles\n",
      (L3L4.data.ring_status_data. Event == IISDNringflgRAW_CADENCE) ?
          "RAW" : "DEGLITCHED",
      (L3L4.data.ring_status_data. RingStatus == IISDNringstatRING_ON) ?
          "ON" : "OFF";
      L3L4.data.ring_status_data, RingCount);
}
```

L3L4mRX_WINK

Description

Notifies the host that a wink signal is received.

Currently, this message is only sent when Feature Group B/Feature Group D is enabled. This is done when the protocol is loaded using *L4L3mENABLE_CAS*:

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.fgb_fgd_mode`

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mRX_WINK)
{
   printf("L3L4mRX_WINK B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mSEIZE_COMP

Description

Notifies the application that the procedure for establishing an outbound call initiated due to an *L4L3mCALL_REQUEST* has finished, and the module starts monitoring the line for an answer condition.

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mSEIZE_COMP)
{
   printf("L3L4mSEIZE_COMP B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mSETUP_IND

Description

Notifies the host of an incoming call. Any call setup information available (DID digits, CallerID and so forth) is returned in this message.

The host can then decide to answer the call (*L4L3mCONNECT_REQUEST*) or reject it (*L4L3mCLEAR_REQUEST*).

Arguments

Data is specified in:

L3_to_L4_struct.data.setup_data

See Also

IISDN_SETUP_DATA, *Calling Party (IISDN_CALLING_PARTY)*, *Called Party (IISDN_CALLED_PARTY)*

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mSETUP_IND)
{
   printf("L3L4mSETUP_IND B-channel %02d:%02d call_ref=%04X"
      "call_type=%d calling_party=%s called party=%s\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref,
      L3L4.data.setup_data.call_type,
      L3L4.data.setup_data.calling_party_digits,
      L3L4.data.setup_data.called_party_digits);
}
```

L3L4mSTATUS_IND

Description

Not currently used. Indicates an internal error in the protocol stack or a signaling fault (occurrence of a line state considered invalid by the protocol).

Arguments

Data is specified in:

L3_to_L4_struct.data.r2_call_status

See Also

IISDN_R2_CALL_STATUS

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mSTATUS_IND)
{
   printf("L3L4mSTATUS_IND B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mTX_HOOKFLASH_END

Description

Notify the host that a hookflash is sent.

The duration of the hookflash is set when the protocol is loaded (*L4L3mENABLE_CAS*):

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data.hookflash_timer`

This can be overridden by using *L4L3mTX_HOOKFLASH*.

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mTX_HOOKFLASH_END)
{
   printf("L3L4mTX_HOOKFLASH_END B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

L3L4mTXWINK_END

Description

Notifies the host that a wink is sent in acknowledgement to an incoming call. This notification is necessary in WINK_START and DELAY_DIAL protocols.

All parameters related to the duration of the wink are set when the protocol is loaded (*L4L3mENABLE_CAS*):

L4_to_L3_struct.
`data.cas_data.cas_params.robbed_bit_data`

dptimer_pre_wink	Delay before transmitting the wink
dptimer_wink	Duration of the wink proper
dptimer_post_wink	Guard time after sending the wink

Arguments

None.

Example

```
L3_to_L4_struct L3L4;

BsmiControlRead (fd, &L3L4);
//check error code here

if (L3L4.msgtype == L3L4mTXWINK_END)
{
   printf("L3L4mTXWINK_END B-channel %02d:%02d call_ref=%04X\n",
      L3L4.lapdid, L3L4.bchannel, L3L4.call_ref);
}
```

26 - Host to Module (L4L3m) Messages

This chapter explains each BSMI L4L3 message a host sends to a Dialogic® Brooktrout® module to provide information about the call.

The L4L3 messages are described in alphabetical order beginning on *page 1006*.

L4L3 messages provide the host processor with a concise, consistent interface for the full range of Dialogic® Brooktrout® modules. The C language structures that make up this interface are contained in the *iisdn.h* files included in the Dialogic® Brooktrout® firmware release.

Note: In the case of discrepancies between the descriptions in the *iisdn.h* file and this document, follow the structures in *iisdn.h*. Changes in the software since the release of this document are found in the Release Notes that accompany the product.

L4L3mALERTING_REQUEST

Purpose

Sends an ALERTING message to the network, indicating that the local terminal (host end) is ringing.

Message

IISDN_AL_CON_DATA *al_con_data*

Message ID

0x83

Input Fields

IISDN_AL_CON_DATA *al_con_data*;

Input

al_con_data

Alerting and Connecting Data Message. See *Alerting and Connecting Data Message (IISDN_AL_CON_DATA)* on *page 851*.

Output

Return values.

After the module successfully processes this message, it sends an ALERTING message to the network. If this is the first response to a received *L3L4mSTATUS_IND* and the B-channel negotiation feature is enabled, the B-channel to be used for this call is included in the message. Additional IEs are appended if they were included in the IISDN_IE_STRUCT.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid *lapdid* value has been used. The only *lapdid* values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errLAPDID_NOT_ESTABLISHED 02

ISDN D-channel identified by the LAP-D ID has not been established.

L3L4errCALL_REF_ERROR 06

Invalid call reference value specified.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

L3L4errINVALID_MSG_FOR_STATE 16

Indicates either the Dialogic® Brooktrout® firmware cannot generate a CALL PROCEEDING message for this call because of its current state (CALL PROCEEDING not a valid message for the call state) or the network is clearing the call. In the latter case, this error message is immediately preceded by an L3L4mCLEAR_REQUEST or an L3L4mCLEAR_WITH_RESTART message.

L3L4errDCHAN_TEMP_UNAVAIL 30

D-channel switchover in progress; applies to NFAS configurations only.

See message details in *L3L4mERROR* on *page 1106*.

Details

In a standard ISDN call scenario, the host issues this message after receiving *L3L4mSETUP_IND*. Because of this, the host specifies the L4 reference value for the call in this message.

If the interface system is using the AT&T 4ESS Fast Connect Feature, no PROGRESS or ALERTING messages are required for call connection.

bchannel and *iface* can be used in support of ISDN B-channel negotiation. If the *L4L3mALERTING_REQUEST* message is the first response to an *L3L4mSETUP_IND* and B-channel negotiation is enabled, *bchannel* and *iface* must be used. In all other cases, these fields are ignored by the module; value of 0x00 is recommended for both.

Use the IE structure in this message to send custom Information Elements as part of the ALERTING message. See *Information Element (IISDN_IE_STRUCT)* on *page 865* for more information.

L4L3mCALL_PROCEEDING_REQUEST

Purpose

Used only if the host application is performing the B-channel negotiation.

Message

IISDN_CALL_PROC_DATA *proc_data*

Message ID

0x89

Input Fields

```
IISDN_Q922_DLCI fr_dlci;
unsigned long bchannel_mask;
unsigned char bchannel;
unsigned char iface;
unsigned char ie_count;
IISDN_PROGRESS_IND progress_ind;
```

Input

fr_dlci

Q.933 DLCI Negotiation Structure. Refer to *Q.933 DLCI Negotiation (IISDN_Q922_DLCI)* on *page 870*.

bchannel_mask

B-channel Bit Mask. Valid only for multi-rate ISDN calls. Specifies the individual IISDN channels used for a multi-rate call. Channels are numbered 1 to 24 (ISDN service in US only) with channel 1 as the least significant bit. First multi-rate channel must also be specified in *bchannel*.

bchannel

B-channel. Identifies the channel to be used for the call. In this message, *bchannel* is used only when B-channel negotiation is enabled by setting *b_chan_negot* in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message. If the feature is not enabled, set *bchannel* to 0x00. Channels are numbered as listed below:

ISDN 23B+DChannels numbered 1 – 23.

ISDN NFASChannels numbered 1 – 24. The *iface* value indicates the span where the channel resides.

ISDN 384KFirst of six channels used for 384K (H0) call. Possible values are 1, 7, 13 for standard ISDN, or 1, 7, 13, and 19 for ISDN NFAS service.

ISDN 1536KMust be channel 1 for 1536K (H11) call.

multi-rate ISDNFirst channel of multi-rate call (set of channels specified in *bchannel_mask*).

iface

Non-Facilities Associated Signaling (NFAS) Interface. Indicates the span where the channel specified in *bchannel* field resides. The host application must maintain the mapping of span (line) to interface. In this message, *iface* is used only when, in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message, the protocol has done both of the following:

- Enabled B-channel negotiation using *b_chan_negot*
- Enabled NFAS using *nfas*

If either of these features are not enabled, *iface* is ignored and set to 0x00. If B-channel negotiation is enabled and NFAS is not in use, set *iface* to 0xFF.

ie_count

Specifies the number of Information Elements (IEs) included in the IE Structure in this message.

progress_ind

Progress Indication Structure. See *Progress Indication (IISDN_PROGRESS)* on *page 867*.

ie

IE Structure. See *Information Element (IISDN_IE_STRUCT)* on *page 865*.

Output

Return values.

After this message has been successfully processed, the module sends a CALL PROCEEDING message to the network. If this is the first response to a received *L3L4mSETUP_IND* and the B-channel negotiation feature is enabled, the B-channel used for this call is included in the message. Additional IEs are appended if included in the IISDN_IE_STRUCT.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid lapdid value has been used. The only lapdid values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errLAPDID_NOT_ESTABLISHED 02

ISDN D-channel identified by the LAP-D ID has not been established.

L3L4errCALL_REF_ERROR 06

Invalid call reference value specified.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

L3L4errINVALID_MSG_FOR_STATE 16

Indicates the Dialogic® Brooktrout® firmware cannot generate a CALL PROCEEDING message for this call because of its current state (CALL PROCEEDING is not a valid message for the call state).

L3L4errDCHAN_TEMP_UNAVAIL 30

D-channel switchover in progress; applies to NFAS configurations only.

See message detail in *L3L4mERROR* on *page 1106*.

Details

Enable B-channel negotiation by setting *b_chan_negot* in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message for this D-channel. When this feature is enabled, the Dialogic® Brooktrout® firmware does not automatically send a CALL PROCEEDING message to the network in response to an incoming SETUP. Instead, the host must use the *L4L3mCALL_PROCEEDING_REQUEST* to instruct the module to generate this message, if the call is to be accepted. Alternately, the host can reject the call at this stage using the *L4L3mCLEAR_REQUEST*.

Use the IE structure in this message to send custom Information Elements as part of the CALL PROCEEDING message. See *Information Element (IISDN_IE_STRUCT)* on *page 865*.

L4L3mCALL_REQUEST

Purpose

Starts an outgoing call. When the interface type is ISDN, the module sends a SETUP message to the network.

Message

IISDN_CALL_REQ_DATA *call_req_data*

Message ID

0x81

Input Fields

```
unsigned long bchannel_mask;
unsigned char preferred;
unsigned char net_spfc;
unsigned char iface;
unsigned long call_type;
IISDN_CALLING_PARTY calling_party;
IISDN_CALLED_PARTY called_party;
IISDN_REDIRECT_NUM redirect_num;
unsigned short override_bc_len;
unsigned char bearer_cap [IISDN_MAX_BC];
IISDN_USER_INFO  user_info;
IISDN_Q922_DLCI  fr_dlci;
unsigned char rlt_service;
unsigned char ie_count;
IISDN_ORIG_CALLED_NUM  orig_called_num;
IISDN_IE_STRUCT ie;
unsigned char reserved;
```

Input

bchannel_mask

B-channel Bit Mask. Valid only for multi-rate ISDN calls. Specifies the individual IISDN channels used for a multi-rate call. Channels are numbered 1 to 24 (ISDN service in US only) with channel 1 as the least significant bit. First multi-rate channel must also be specified in *bchannel*.

bchannel

B-channel. Identifies the channel to be used for the call. In this message, *bchannel* is used only when B-channel negotiation is enabled by setting *b_chan_negot* in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message. If the feature is not enabled, set *bchannel* to 0x00. Channels are numbered as listed below:

ISDN 23B+DChannels numbered 1 – 23.

ISDN NFASChannels numbered 1 – 24. The *iface* value indicates the span where the channel resides.

ISDN 384KFirst of six channels used for 384K (H0) call. Possible values are 1, 7, 13 for standard ISDN, or 1, 7, 13, and 19 for ISDN NFAS service.

ISDN 1536KMust be channel 1 for 1536K (H11) call.

multi-rate ISDNFirst channel of multi-rate call (set of channels specified in *bchannel_mask*).

preferred

Preferred channel. Indicates whether the B-channels specified must be used for this call. If the feature is not enabled, set to 0x00. Possible values include:

Specified channel must be used (exclusive).	0x00
Network can use another channel (preferred); B-channel negotiation.	0x01

net_spfc

Call-by-Call Feature. Specifies the network-specific facility used. Generally, the non-specific default value is used. If another value is specified, the PRI line purchased from the service provider must have been configured to support that call type. Possible values include:

IISDNnsNULL 0x00

Default value; no network-specific facility Information Element (IE) is encoded.

IISDNnsATT_SDN or IISDNnsNTI_PRIVATE 0x01

AT&T Software Defined Network or Nortel Private Network.

IISDNnsATT_MEGACOM or IISDNnsNTI_OUTWATS 0x02

AT&T Megacom or Nortel OutWATS.

IISDNnsNTI_FX 0x03

Nortel Foreign Exchange.

IISDNnsNTI_FX 0x04

Northern Telecom Foreign Exchange.

IISDNnsNTI_TIE_TRUNK 0x05

Northern Telecom Tie Trunk.

IISDNnsATT_ACCUNET 0x06

AT&T Accunet.

IISDNnsATT_I800 0x08

AT&T International 800 Service.

IISDNnsATT_MULTIQUEST *or* IISDNnsNTI_TRO 0x10

Northern Telecom TRO Call.

iface

Non-Facilities Associated Signaling (NFAS) Interface. Indicates the span where the channel specified in *bchannel* field resides. The host application must maintain the mapping of span (line) to interface. In this message, *iface* is used only when, in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message, the protocol has done both of the following:

- Enabled B-channel negotiation using *b_chan_negot*
- Enabled NFAS using *nfas*

If either of these features are not enabled, *iface* is ignored and set to 0x00. If B-channel negotiation is enabled and NFAS is not in use, set *iface* to 0xFF.

call_type

Call Type. Bit mask identifying what type of call to establish. Call types are constructed using the following values:

IISDNcalltypVOICE 0x00000001

Normal voice call in North America (µ-law); most common value.

IISDNcalltypMODEM 0x00000002

3.1kHz audio in North America (µ-law).

IISDNcalltyp56K 0x00000004

56K data call, unknown type.

IISDNcalltyp64K 0x00000008

64K data call, unknown type.

IISDNcalltyp64K_REST 0x00000010

Restricted 64K data call.

IISDNcalltyp384K 0x00000020

384K (H0) data call, unknown type.

IISDNcalltyp384K_REST 0x00000040

Restricted 384K (H0) data call.

IISDNcalltyp64K_V110 0x00002000

64K V.110 data call.

IISDNcalltypmulti-rate_DATA 0x00004000

ISDN multi-rate service call of n x 64K channels, unrestricted.

IISDNcalltyp1536K 0x00008000

1536K (H11) data call, unrestricted.

IISDNcalltyp56K_UNREST 0x00010000

56K data call, unrestricted.

IISDNcalltypALAW_VOICE 0x00020000

Voice call outside of North America (A-law).

IISDNcalltypALAW_MODEM 0x00400000

3.1kHz audio call outside of North America (A-law).

IISDNcalltypULAW_7KHZ 0x00800000

7 kHz call in North America (µ-law).

IISDNcalltypALAW_7KHZ 0x00100000

7 kHz call outside of North America (A-law).

calling_party

Calling Party Structure. *See* *Calling Party (IISDN_CALLING_PARTY)* on *page 857*.

called_party

Called Party Structure. See *Called Party (IISDN_CALLED_PARTY)* on *page 855*.

redirect_num

Redirecting Number Structure. See *Redirecting Number (IISDN_REDIRECT_NUM)* on *page 871*.

override_bc_len

Override B-bearer Capability. Specifies the length of the *bearer_cap* array below. Recommended setting is 0x00 (no array included).

bearer_cap [IISDN_MAX_BC]

Bearer Capability Data. An array used to create a custom call_type. The length of the array is specified in *override_bc_len*. Because the *call_type* settings specified previously in the message include most call types possible, use of this capability is not recommended.

user_info

User Info Structure. See *User Info (IISDN_USER_INFO)* on *page 875*.

fr_dlci

Q.933 DLCI Negotiation Structure. See *Q.933 DLCI Negotiation (IISDN_Q922_DLCI)* on *page 870*.

rlt_service

Release Trunk Service. This feature is used for interfacing with DMS-250. The On function sends a facility IE in the setup message.

0x00= off

0x01= on

ie_count

IE Count. Specifies the number of IEs included in the IE Structure in this message.

orig_called_num

Specifies the appropriate number of digits. A value of 0 designates the feature is unavailable.

ie

IE Structure. See *Information Element (IISDN_IE_STRUCT)* on *page 865*.

reserved

1 Allows this call to be made on a channel that already has a call in the auxiliary HOLD state.

Output

Return values.

ISDN Q.931 Calls

After the module successfully processes this message, it initiates an outgoing call by sending a SETUP message to the network. An *L3L4mPROGRESS*, *L3L4mALERTING*, or *L3L4mCONNECT* message is returned to the host to indicate the network is processing the call. If the call is rejected by the network, the host receives either an *L3L4mCLEAR_REQUEST* or *L3L4mCLEAR_WITH_RESTART_REQUEST* message.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid *lapdid* value has been used. The only *lapdid* values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errLAPDID_NOT_ESTABLISHED 02

ISDN D-channel identified by the LAP-D ID has not been established.

L3L4errINVALID_CALLED_NUMBER 03

For ISDN calls, Called Party number is either invalid (not ASCII digits 0 thru 9) or longer than 24 digits.

L3L4errNO_CRSTRUCT_AVAILABLE 05

No call record structures are available to start an outgoing call.

L3L4errINVALID_B_CHANNEL 07

B-channel specified is invalid.

L3L4errB_CHANNEL_RESTARTING 08

For ISDN calls, the B-channel is in the progress of restarting.

L3L4errINVALID_CALL_TYPE 10

The value specified for the *call_type* field is invalid.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

L3L4errINVALID_INTERFACE 19

Interface number specified is not between 0 an 19 (NFAS configurations).

L3L4errDCHAN_TEMP_UNAVAIL 20

D-channel switchover in progress; applies to NFAS configurations only.

L3L4errB_CHANNEL_INUSE 30

B-channel specified is already involved in a call or an incoming SETUP message has been received specifying this B-channel and its preferred bit is cleared.

L3L4errDLCI_MANDATORY 48

Indicates the message was rejected because the L3L4 common header did not specify a DLCI. A DLCI is required for LAP-D data connections.

See message details in *L3L4mERROR* on *page 1106*.

Details

The IE structure can be used in this message to send custom Information Elements as part of the SETUP message. See *Information Element (IISDN_IE_STRUCT)* on *page 865* for more information.

L4L3mCLEAR_REQUEST

Purpose

Clears (tears down) a call or refuses an incoming call.

Message

IISDN_CLR_DATA *clr_data*

Message ID

0x85

Input Fields

```
IISDN_CAUSE cause;
IISDN_USER_INFO user_info;
IISDN_PROGRESS_IND progress_ind;
unsigned char ie_count;
IISDN_IE_STRUCT ie;
```

Input

cause

Cause Data Structure. See *Cause Data (IISDN_CAUSE)* on *page 860*.

user_info

User Info Structure. See *User Info (IISDN_USER_INFO)* on *page 875*.

progress_ind

Progress Indication Structure. See *Progress Indication (IISDN_PROGRESS)* on *page 867*.

ie_count

IE Count. Specifies the number of Information Elements (IEs) included in the IE Structure in this message.

ie

IE Structure. See *Information Element (IISDN_IE_STRUCT)* on *page 865*.

Output

After the module successfully processes this message, it sends the appropriate message (DISCONNECT, RELEASE, or RELEASE COMPLETE) to the network and an *L3L4mCLEAR_REQUEST* to the host. Upon receipt of the *L3L4mCLEAR_WITH_RESTART_REQUEST*, *L3L4mCLEAR_REQUEST*, or an *L3L4mDISCONNECT* message indicating all_calls_dropped, the B-channel is available for a new call.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid *lapdid* value has been used. The only *lapdid* values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errLAPDID_NOT_ESTABLISHED 02

ISDN D-channel identified by the LAP-D ID has not been established.

L3L4errCALL_REF_ERROR 06

Invalid call reference value specified or module has already cleared the call. In the latter case, this message is immediately preceded by an *L4L3mCLEAR_REQUEST* message.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

L3L4errB_CHANNEL_INUSE 30

B-channel specified is already involved in a call, or an incoming SETUP message has been received specifying this B-channel and its *preferred* bit is cleared.

See message details in *L3L4mERROR* on *page 1106*.

Details

This message applies to all ISDN calls and the structure is identical to that of the following messages:

- *L3L4mDISCONNECT*
- *L3L4mCLEAR_REQUEST*

The module *always* generates either an *L3L4mCLEAR_REQUEST* or *L3L4mCLEAR_WITH_RESTART_REQUEST* when a call is cleared. The B-channel cannot be used for a new call until this L3L4 message is acknowledged/received by the host.

Note: The only exception to the above is in the ISDN case where a D-channel is restarted and all calls associated with it are dropped. In this case, the setting of *all_calls_dropped* in the *L3L4mDISCONNECT* message indicates if the protocol restarts the B-channels associated with the specified D-channel and drops the calls.

You can also use the *L4L3mCLEAR_REQUEST* message to clear a call initiated by the host before a response is received from the network. In this case, generate this message with the L4 reference value used in the *L4L3mCALL_REQUEST* and a call reference value of zero. The module then begins the call clearing processes required by the network.

Use the IE structure in this message to send custom Information Elements as part of the ISDN message. See the structure and its use described in *Information Element (IISDN_IE_STRUCT)* on *page 865*.

L4L3mCONNECT_REQUEST

Purpose

Sends a CONNECT message to the network to inform it that an incoming call was answered.

Message

IISDN_AL_CON_DATA *al_con_data*

Message ID

0x84

Input Fields

IISDN_AL_CON_DATA *al_con_data*;

Input

al_con_data

Alerting and Connecting Data Message. See *Alerting and Connecting Data Message (IISDN_AL_CON_DATA)* on *page 851*.

Output

Return values.

After the module successfully processes this message, it sends a CONNECT message to the network.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid *lapdid* value has been used. The only *lapdid* values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errLAPDID_NOT_ESTABLISHED 02

ISDN D-channel identified by the LAP-D ID has not been established.

L3L4errCALL_REF_ERROR 06

Invalid call reference value specified.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

L3L4errINVALID_MSG_FOR_STATE 16

Indicates the Dialogic® Brooktrout® firmware cannot generate a CALL PROCEEDING message for this call because of its current state (CALL PROCEEDING not a valid message for the call state).

L3L4errDCHAN_TEMP_UNAVAIL 30

D-channel switchover in progress; applies to NFAS configurations only.

L3L4errDLCI_MANDATORY 48

Indicates the message was rejected because the L3L4 common header did not specify a DLCI. A DLCI is required for LAP-D data connections.

See message details in *L3L4mERROR* on *page 1106*.

Details

You can send the *L4L3mCONNECT_REQUEST* message in response to an *L3L4mSETUP_IND* when connected to a service using the AT&T 4ESS Fast Connect Feature. When used in this manner and if the B-channel negotiation feature is enabled, use *bchannel* and *iface*. In all other cases, these fields are ignored by the module; value of 0x00 is recommended for both.

L4L3mDISABLE_B_CHANNEL

Purpose

Generates ISDN SERVICE messages (North American PRI only) for ATT B Channel Maintenance protocol only.

Message

IISDN_BCHANNEL_ID *channel*

Message ID

0xA3

Input Fields

```
unsigned long bchannel_mask;
unsigned char bchannel;
unsigned char iface;
unsigned char busy_out_chan;
unsigned char use_bit_mask;
unsigned long n_bchannel;
```

Input

bchannel_mask

B-channel Bit Mask. Valid only for multi-rate ISDN calls. Specifies the individual PRI channels used for a multi-rate call. Channels are numbered 1 to 24 (ISDN service in US only) with channel 1 as the least significant bit. First multi-rate channel must also be specified in *bchannel*.

0 = not used

1 = multi-rate call

bchannel

B-channel. Identifies the channel unless *use_bit_mask* is nonzero and NFAS is used. Channels are numbered as listed below:

ISDN 23B+DChannels numbered 1 – 23.

ISDN NFASChannels numbered 1 – 24. The *iface* value indicates the span where the channel resides.

ISDN 384KFirst of six channels used for 384K (H0) call. Possible values are 1, 7, 13 for standard ISDN, or 1, 7, 13, and 19 for ISDN NFAS service.

ISDN 1536KMust be channel 1 for 1536K (H11) call.

multi-rate ISDNFirst channel of multi-rate call (set of channels specified in *bchannel_mask*).

iface

Non-Facilities Associated Signaling (NFAS) Interface. Indicates the span where the channel specified in *bchannel* field resides. The host application must maintain the mapping of span (line) to interface. In this message, *iface* is used only when, in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message, the protocol has done both of the following:

- Enabled B-channel negotiation using *b_chan_negot*
- Enabled NFAS using *nfas*

If either of these features are not enabled, *iface* is ignored and set to 0x00. If B-channel negotiation is enabled and NFAS is not in use, set *iface* to 0xFF.

busy_out_chan

Not used.

use_bit_mask

Use Bit Mask. Specifies use of *n_bchannel* to indicate the B-channels for the generated message; allowed for NFAS configurations only.

0x00 = use *bchannel*

0x01 = use *n_bchannel*

n_bchannel

NFAS B-channel. This field is ignored if *cnfg.q931.nfas* is not set in a previous *L4L3mENABLE_PROTOCOL*. Bit mask identifying which B-channels to activate on this interface. T1 B-channels are numbered 1 - 24 (24 is usually reserved for the D-channel). Typical setting is 0x00FFFFFE.

1 = enable B-channel

0 = disable B-channel

Output

Return values.

ISDN Q.931 Protocol

After the module successfully processes this message, it places the B-channels into near-end out-of-service state and sends a SERVICE (OOS) message to the network. When the network responds with a SERVICE ACK message, the module generates an *L3L4mB_CHANNEL_STATUS* with a *b_channel_data* set to 0x00 IISDNbcsOUT_OF_SERVICE report indicating the change of state. The B-channel cannot be used for calls until it is brought back into service using the *L4L3mENABLE_B_CHANNEL* message.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_NOT ESTABLISHED 02

Q931 is not currently established on the *lapdid*.

L3L4errINVALID_B_CHANNEL 07

IISDN_BCHANNEL_ID.*bchannel* is not in the range 1-24.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

L3L4errINVALID_INTERFACE 19

NFAS *iface* value is not recognized.

See message details in *L3L4mERROR* on *page 1106*.

Details

When the Q.931 protocol stack has been enabled, this message causes the module to send a SERVICE message to the connected network equipment with a status of out of service. The B-channel affected is specified in the structure IISDN_BCHANNEL_ID. This message is not meant to reserve channels; frequent SERVICE messages for a B-channel might be interpreted as a fault condition by the connected switching equipment.

When NFAS has been enabled, the B-channels can be specified using *n_bchannel*. This bit mask indicates the state (in service or out-of-service) of each channel on the interface. This bit mask can be used to specify either a single or an entire interface of B-channels.

L4L3mDISABLE_PROTOCOL

Purpose

Disables the protocol stack running on the HDLC channel specified in the common header. Specifies the HDLC channel using the LAP-D ID and LLI common header bytes.

Message

None.

Message ID

`0xA1`

Input

Supplies all data for this message in the L4L3 common header.

Output

Return values.

After successfully processing this message, the module disables the protocol stack and clears all calls associated with the LAP-D ID and LLI specified in the L4L3 common header. When the protocol is disabled, the module generates an *L3L4mDISCONNECT* report with a status of Not Established.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid *lapdid* value has been used. The only *lapdid* values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errLAPDID_NOT_ESTABLISHED 02

The LAP-D ID for the message signifies that the LAPD is not currently up.

L3L4errINVALID_B_CHANNEL 07

The LAP-D ID message signifies that the B-channel in the CALL REQ is bad.

L3L4errINVALID_LLI 21

The LAP-D ID for which the message was issued is not currently established.

See message details in *L3L4mERROR* on *page 1106*.

L4L3mENABLE_B_CHANNEL

Purpose

Generates an ISDN SERVICE message for ATT B-channel Maintenance protocol only.

Message

IISDN_BCHANNEL_ID *channel*

Message ID

0xA2

Input Fields

```
unsigned long bchannel_mask;
unsigned char bchannel;
unsigned char iface;
unsigned char busy_out_chan;
unsigned char use_bit_mask;
unsigned long n_bchannel;
```

Input

bchannel_mask

B-channel Bit Mask. Valid for multi-rate ISDN calls only. Specifies the individual PRI channels used for a multi-rate call. Channels are numbered 1 to 24 (ISDN service in US only) with channel 1 as the least significant bit and 24 as the most significant bit. *bchannel* must specify the first multi-rate channel.

0 = not used

1 = multi-rate call

bchannel

B-channel. Identifies the channel unless *use_bit_mask* is a nonzero value and NFAS is used. Channels are numbered as listed below:

ISDN 23B+DChannels numbered 1 – 24.

ISDN NFASChannels numbered 1 – 24. The *iface* value indicates the span where the channel resides.

iface

Non-Facilities Associated Signaling (NFAS) Interface. Indicates the span where the channel specified in *bchannel* field resides. The host application must maintain the mapping of span (line) to interface. In this message, *iface* is used only when, in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message, the protocol has done both of the following:

- Enabled B-channel negotiation using *b_chan_negot*
- Enabled NFAS using *nfas*

If either of these features are not enabled, *iface* is ignored and set to 0x00. If B-channel negotiation is enabled and NFAS is not in use, set *iface* to 0xFF.

busy_out_chan

Not used.

use_bit_mask

Use Bit Mask. Specifies use of *n_bchannel* to indicate the B-channels for which the message is to be generated; allowed for NFAS configurations only.

0x00 = use *bchannel*

0x01 = use *n_bchannel*

n_bchannel

NFAS B-channel. The *n_bchannel* field is ignored if *cnfg.q931.nfas* is not set in a previous *L4L3mENABLE_PROTOCOL*. The Bit Mask identifies which B-channels to activate on this interface. The T1 B-channels are numbered 1 - 24 (24 is usually reserved for the D-channel). Typical setting is 0x00FFFFFE.

1 = enable B-channel

0 = disable B-channel

Output

Return values.

ISDN Q.931 Protocol

After successfully processing this message, the Dialogic® Brooktrout® module sends one or more SERVICE messages to the network with a status of in service for each B-channel represented in the message.

When the network responds with a SERVICE ACK message, the module generates an *L3L4mB_CHANNEL_STATUS* report with the appropriate *b_channel_data* value, indicating the receipt of the message from the network.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_NOT_ESTABLISHED 02

The LAP-D ID for the message signifies that the LAPD is not currently up.

L3L4errINVALID_B_CHANNEL 07

The LAP-D ID message signifies that the B-channel in the CALL REQ is bad.

L3L4errINVALID_INTERFACE 09

NFAS.*iface* value is not recognized.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

See message details in *L3L4mERROR* on *page 1106*.

Details

When the application enables the Q.931 protocol stack, this message causes the module to send a SERVICE message to the connected network equipment with a status of in service. The IISDN B-channel structure specifies the affected IISDN_BCHANNEL_ID.

Specify the B-channel using *n_bchannel* when NFAS is enabled. This bit mask indicates the state (in service or out-of-service) of each channel on the interface. This bit mask can be used to specify either a single or an entire interface of B-channels.

L4L3mENABLE_PROTOCOL

Purpose

Specifies and enables Layer 1, Layer 2, and Layer 3 processing for a specific LAP-D ID.

Message

IISDN_ENA_PROTO_DATA *enable_protocol*

Message ID

0xB6

Input Fields

```
unsigned long command;      /* command mode*/

unsigned char l1_mode;      /* IISDN_LEVEL1_CNFG*/
rate_adapt
   unsigned char enable;
   unsigned char rate_adapt_value;
   short pad;

unsigned char l2_mode;      /* IISDN_LEVEL2_CNFG*/
unsigned char dce_dte;
unsigned char no_sabme;
unsigned char L2_detail;
unsigned char priority;
unsigned char no_reestab;
unsigned char mode_TEI_1;
unsigned char no_piggyback;
unsigned char l2_const;
unsigned char TEI_mode;

unsigned char l3_mode;      /* IISDN_LEVEL3_CNFG*/
unsigned char pad;
unsigned short pad1;
IISDN_Q931_CNFG q931_cnfg;
unsigned short switch_type;
unsigned short variant;
unsigned long call_filtering;
unsigned short jate_redial_method;
IISDN_Q931_TIMERS q931_timers;
unsigned long
   bchannel_service_state[IISDN_NUM_DS1_INTERFACES];
unsigned char nfas;
unsigned char net_side_emul;
unsigned char b_chan_negot;
unsigned char proc_on_exclusv;
unsigned char chanid_slot_map;
```

```
unsigned char sprs_chanid_callproc;
unsigned char no_chanid_callproc;
unsigned char append_raw_qmsg;
unsigned char ccitt_mode;
unsigned char raw_qmsg;
unsigned char no_ie_errcheck;
unsigned char user_ie_encode;
unsigned char send_I3I4_callproc;
unsigned char sending_cmplt;
unsigned char report_incoming_callproc;
unsigned char no_tx_conn_act;
unsigned char no_rx_conn_act;
unsigned char sprs_chanid_setupack;
unsigned char no_chanid_setupack;
unsigned char no_canned_spid_rej;
unsigned char call_reject_notify;
unsigned char primary_lapdid;
unsigned char primary_ifnum;
unsigned char subscribe_connack;
unsigned char basic_rate;
unsigned char suppress_auto_spid;
unsigned char spid_len;
unsigned char spid_1_len;
unsigned char dn_len;
unsigned char dn_1_len;
char spid[IISDN_MAX_SPID_LEN];
char spid_1[IISDN_MAX_SPID_LEN];
char dn[IISDN_MAX_DN_LEN];
char dn_1[IISDN_MAX_DN_LEN];
```

Input

command

Enable Protocol Command Mode. Specifies how Dialogic® Brooktrout® firmware processes this message. This mode provides a mechanism to delivering commands to a previously configured protocol. Values require that a previous *L4L3mENABLE_PROTOCOL* message establishing a Q.921/LAP-D DLCI was already successfully processed. Possible values include:

IISDNepcmdNO_COMMAND 0x00000000

Normal processing; Dialogic® Brooktrout® firmware interprets all structures in this message and configures the protocol stack accordingly. Default value.

IISDNepcmdDL_ESTABLISH 0x00000001

Used with Layer 2 protocols only. Specifies sending a Layer 2 SABME (DL-ESTABLISH) message. No other structures are processed by the module when this command is sent.

IISDNepcmdDL_RELEASE 0x00000002

Used with Layer 2 protocols only. Specifies sending a Layer 2 DISC (DL-RELEASE) message. No other structures are processed by the module when this command is sent.

l1_mode

Level 1 Mode. Specifies the Layer 1 mode of operation for this channel.

IISDNl1modHDLC 0x00

HDLC packetization mode; default value.

rate_adapt

Rate Adaption Structure. Specifies if rate adaption is used on this channel and the bandwidth of the channel.

enable

Enable Rate Adaption. Specifies if rate adaption is enabled on this channel.

0x00 = rate adaption disabled

0x01 = rate adaption enabled

rate_adapt_value

Rate Adaption Value. Specifies type of rate adaption used. Default value is 0x7F (for standard 56 K rate adaption).

pad

Pad to preserve longword alignment; set to 0x0000.

l2_mode

Level 2 mode. Specifies the Layer 2 mode of operation for this channel.

IISDNl2modLAP_D 0x00

Enable Q.921 data link layer protocols; default value.

IISDNl2modLAP_D_EFA 0x06

Q.921 data link layer w/Envelope Function Address (4 bytes).

dce_dte

DCE/DTE. Specifies the signaling used by the channel based on the module location (customer premise or network).

IISDNdirUSER_SIDE 0x00

Configures the channel for user side signaling (customer premise equipment) for Q.931

IISDNdirNETWORK_SIDE 0x01

Configures the channel for network side signaling for Q.931.

no_sabme

No SABME Feature. Establishes Layer 2 processing without transmitting a SABME message until a SABME is received. Generally used when connecting two modules back-to-back to establish the link. The link can be better transitioned from TEI-assigned to multiframe-established state by issuing an IISDNepcmdDL_ESTABLISH.

0x00 = feature disabled

0x01 = feature enabled

L2_detail

Layer 2 Detail Mode. Reports all L2 errors to the control interface via the *L3L4mDISCONNECT* message.

0x00 = detail mode disabled

0x01 = detail mode enabled

priority

Transmission Priority. Assigns a transmission priority to the channel; priorities range from 0 (lowest) to 255 (highest). If two LLIs on a single physical HDLC stream have the same priority setting, the channel enabled first has the higher priority.

no_reestab

No Re-establishment. Prevents channel reestablishment when the channel receives or transmits a SABME message once the Layer 2 connection has been established.

0x00 = reestablish channel

0x01 = do not reestablish channel

mode_TEI_1

Custom TEI (Terminal Endpoint Indicator) Mode. Enables asymmetrical TEI assignment when passing Layer 2 and Layer 3 messages over FDL channels.

0x00 = standard LAP-D TEI assignment
0x01 = asymmetrical TEI assignment

no_piggyback

Acknowledge All Messages. Causes the module to acknowledge all messages with an RR (receiver ready). This mode introduces additional traffic overhead and is typically used only when performing protocol conformance testing.

l2_const

Level 2 Configuration. Configures Layer 2 protocol timers.

TEI_mode

TEI negotiated (auto TEI assignment). Value = 1 (see *mode_TEI_1* above). Usually used with BRI in a point-to-multipoint connection.

l3_mode

Level 3 mode. Specifies the Layer 3 mode of operation for this channel. Possible values are:

IISDNl3modDISABLED 0x00

LAP D Disabled message notice.

IISDNl3modQ931 0x01

ITU-T Q.931 call control protocol.

pad

Pad to preserve longword alignment; set to 0x00.

pad1

Pad to preserve longword alignment; set to 0x0000.

q931_cnfg

IISDN_Q931_CNFG structure. Valid only when *l3_mode* is set to IISDNl3modQ931. Defines the operating characteristics for Q.931 applications. This structure is described below:

switch_type

Switch Type. Specifies the type of equipment connected to this span. This setting works in combination with *variant*; refer to *Table 23 on page 1051* for the supported combinations of these fields. Possible *switch-type* values include:

IISDNstATT_4ESS 0x0000

AT&T 4ESS switch; default value.

IISDNstATT_5ESS 0x0001

AT&T 5ESS switch.

IISDNstNTI_DMS100 0x0002

Nortel DMS-100 switch.

IISDNstNTI_DMS250 0x0003

Nortel DMS-250 switch.

IISDNstMD110_T1 0x0004

Ericsson MD-110 switch (U.S.).

IISDNstMD110_E1 0x0005

Ericsson MD-110 switch (international).

IISDNstSIEMENS 0x0006

Siemens EWSD switch (North American).

IISDNstUNKNOWN 0x0008

Unknown switch that conforms to ITU-T standards.

variant

Q.931 Variant. Indicates the variation of Q.931 used on this span. This setting works in combination with *switch_type*. Refer to *Table 23 on page 1051*, for the supported combinations of these fields. Possible values include:

IISDNvarATT_CUSTOM 0x0000

AT&T as defined in AT&T PUB 41449; default value.

IISDNvarNTI_CUSTOM 0x0001

Northern Telecom as defined in NIS A211-1.

IISDNvarNET3 0x0005

IISDNvarCTR3

For BRI connections throughout Europe. Choosing this variant changes the Layer 2 protocol parameters to their appropriate NET-3 defaults.

IISDNvarNET5 0x0006

IISDNvarCTR4

NET-5 standard for PRI connections throughout Europe (also referred to as Euro-ISDN). Choosing this variant changes the Layer 2 protocol timers to their appropriate NET-5 defaults.

IISDNvar1TR6_IISDN 0x0008

1TR6 standard for PRI connections in Germany.

IISDNvarVN3 0x0009

VN3 standard for France.

IISDNvarCCITT 0x000A

General ITU-T Q.931 conformance.

IISDNvarTS014 0x000D

AUSTEL Technical Standard 014 for PRI connections in Australia.

call_filtering

Call Filtering Bit Mask. Reject incoming call if it does not match the bit settings for accepted call types. Call requests received for this span that do not match the bit setting are denied due to incompatible destination. Set this bit mask to 0x00000000 if no call filtering is performed. Accepted call types are constructed using the following values:

IISDNcalltypVOICE 0x00000001

Normal voice call in North America (µ-law).

IISDNcalltypMODEM 0x00000002

3.1kHz audio in North America (µ-law).

IISDNcalltyp56K 0x00000004

56K data call, unknown type.

IISDNcalltyp64K 0x00000008

64K data call, unknown type.

IISDNcalltyp64K_REST 0x00000010

Restricted 64K data call.

IISDNcalltyp56K_UNREST 0x00010000

56K data call, unrestricted.

IISDNcalltypALAW_VOICE 0x00020000

Voice call outside of North America (A-law).

IISDNcalltypALAW_MODEM 0x00040000

3.1kHz audio call outside of North America (A-law).

IISDNcalltypULAW_7KHZ 0x00080000

7 kHz call in North America (µ-law).

IISDNcalltypALAW_7KHZ 0x00100000

7 kHz call outside of North America (A-law).

jate_redial_method

Specifies the redial restriction method when the country code is JAPAN. Any other value specified in *jate_redial_method* default to IISDN_JATE_REDIAL_2IN3_MINS. Redial restrictions include:

IISDN_JATE_REDIAL_3IN3_MINS 0

Redial restriction applies after 3 call attempts until 3-minute timer expires.

IISDN_JATE_REDIAL_15X 1

The redial restriction applies after 15 call attempts.

IISDN_JATE_REDIAL_NO_RESTRICT 2

No redial restriction applies.

q931_timers

Q.931 timers. *IISDN_Q931_TIMERS structure*. Configure Q.931 Layer 3 protocol timers; timers are specified in 100 ms ticks.

b_channel_service_state [IISDN_NUM_DS1_INTERFACES]

Q.931 Bit Mask. Bit mask identifies which B-channels to activate on each interface.

NFAS based application

- T1 B-channels numbered 1-24
- Interfaces numbered from 0-19

You can bring all the B-channels into service as soon as Layer 3 is initialized by filling in all the desired channels using *b_channel_service_state* and specifically leaving the LSB of that parameter set to 0 (this is default behavior). Setting the LSB to 1 causes only the loading of *b_chan_req* bits (as seen via *L3L4mDISCONNECT*) and not the service state of each B-channel, allowing the host program to bring each individual B-channel into service at a later time.

Q.931 based application

Used for B-channel maintenance in Q.931 applications only; bit mask identifies which B-channels to activate (initialize state as IN SERVICE) on each interface.

- To enable a B-channel, set the corresponding bit in the proper array position to 1.
- To disable a B-channel, set the bit in the proper array position to 0.

T1 B-channels are numbered 1 - 23 (24 is typically reserved for D-channel signaling);

To enable all 23 B-channels set the corresponding interface bit mask to: 0xFFFFFE (bit 0 and high bits are ignored). If using Australian or 1TR6, the variant is 0x7FFFFFFE and the channels are numbered 1 - 30.

If in *L4L3ENABLE_PROTOCOL*, *switch_type* is set to IISDNstATT_5ESS or IISDNstNTI_DMS250 and *variant* is set to IISDNvarNATL_ISDN_1 or IISDNvarNATL_ISDN_2, then the module only loads *b_chan_req* bits and the service state of each B-channel is set to OOS (Out Of Service).

nfas

NFAS. Configure for Non-Facility Associated Signaling (NFAS) operation.

0x00 = NFAS disabled

0x01 = NFAS enabled

net_side_emul

Network side emulation. Emulate network side signaling on the channel.

0x00 = emulation disabled

0x01 = emulation enabled

b_chan_negot

B-channel negotiation. Configure for B-channel negotiation on outgoing calls. CALL PROCEEDING message is not automatically generated to the network following an incoming SETUP message unless the *proc_on_exclusv* bit is also set.

proc_on_exclusv

Proceed on exclusive. Used in conjunction with B-channel negotiation, specifies answering an incoming SETUP message from the network with a CALL PROCEEDING (typical operation) when the network indicates the B-channel specified must be used (exclusive).

chanid_slot_map

Channel ID slot map. Encode Channel ID IE using slot map instead of the channel number.

0x00 = use channel numbers

0x01 = use slot map number

sprs_chanid_callproc

Suppress Channel ID in CALL PROCEEDING. Ignore Channel ID IEs received in incoming CALL PROCEEDING messages.

0x00 = use Channel ID IEs

0x01 = ignore Channel ID IEs

no_chanid_callproc

No Channel ID in CALL PROCEEDING. Allows channel to process a received CALL PROCEEDING message even if the message does not contain a Channel ID IE.

0x00 reject CALL PROCEEDING message without Channel ID IE.

0x01 accept CALL PROCEEDING message without Channel ID IE.

append_raw_qmsg

Append Raw Q.931 Message. Specifies to follow all L3L4 Q.931 messages (such as an *L3L4mSETUP_IND* message) with an *L3L4mRAW_QDATA* message containing the undecoded Q.931 packet. Refer to *L3L4mRAW_QDATA* on *page 1126*, for more information.

0x00 = feature disabled

0x01 = append L3L4mRAW_QDATA message

ccitt_mode

ITU-T Mode. Applies only when variant = IISDNvarCCITT, IISDNvarNET3, or IISDNvarNET5.

Specifies that the connections must strictly adhere to ITU-T Q.931 recommendations (as opposed to Q.931 variations specific to AT&T, Nortel Telecom, and so forth).

0x00 = mode disabled

0x01 = mode enabled

raw_qmsg

Pass Raw Q.931 Message Only. Specifies replacing the standard contents of L3L4 Q.931 call control messages with a *data.raw_q931_data* structure containing the undecoded Q.931 packet.

0x00 = feature disabled

0x01 = replace message contents with undecoded Q.931 packet

no_ie_errcheck

No IE Error Checking. Specifies disabling error checking on IEs received from the network.

0x00 = perform error checking

0x01 = no error checking

user_ie_encode

User IE Encoding. Specifies that IE strings supplied by the host contain all IEs to be transmitted. When this feature is enabled, ISDN does not automatically include mandatory IEs when transmitting Q.931 messages.

0x00 = feature disabled

0x01 = host must supply all IEs to transmit

send_I3I4_callproc

Sends an *L3L4mCALL_PROC_SENT* message after transmitting of a call proceeding.

sending_cmplt

Inserts a sending-complete information element in all outgoing call setup messages, indicating that the called party information is complete, and sends no further called party information.

report_incoming_callproc

This reports the receipt of a call proceeding as an *L3L4mCALL_PROCEEDING*.

no_tx_conn_act

Do not send Connect Acknowledge statement.

no_rx_conn_act

Do not wait for Connect Acknowledge statement.

sprs_chanid_setupack

Suppresses Channel ID info element in the SETUP_ACK message.

no_chanid_setupack

Accepts missing Channel ID info element in the SETUP_ACK message.

no_canned_spid_rej

Passes all 5ESS Custom spid MGMT.INF messages as *L3L4mUNIVERSAL* message. Then sends a canned reject message to the net.

call_reject_notify

Sends *L3L4mCALL_REJECT* messages for incoming calls on OOS channels.

primary_lapdid

NFAS Primary Lap-D ID. Specifies using the HDLC controller channel as the D-channel in an NFAS configuration. Current software supports a value of 0x00 in this field. *primary_lapdid* is ignored. If you do not set the NFAS bit the current software does not support backup D-channels. Therefore, as a configuration value, set *backup_lapdid* and *backup_ifnum* to 0xFF.

primary_ifnum

Primary Interface Number. Specifies the interface on which the primary D-channel resides. Interfaces are numbered from 0 to 19. If you do not set the NFAS bit the *primary_ifnum* is ignored.

subscribe_connack

Indicates that the *L3L4mCONN_ACK_IND* message is received.

basic_rate

1 = Set this line to the BRI protocol

suppress_auto_spid

1 = Suppress sending the SPID for BRI

spid_len

Basic Rate National ISDN-1 SPID. Value: 3 - 20.

spid_1_len

DMS-100 for second phone call.

dn_len

Length of directory number.

dn_1_len

Length of directory number.

spid[IISDN_MAX_SPID_LEN]

Service Profile ID (ASCII).

spid_1[IISDN_MAX_SPID_LEN]

DMA-100 for second phone call.

dn[IISDN_MAX_DN_LEN]

ASCII directory number for *spid*.

dn_1[IISDN_MAX_DN_LEN]

ASCII directory number for *spid_1*.

Output

Return values.

After successfully processing this message, the module brings the D-channel into service as specified. When the D-channel is established, the module generates an *L3L4mDISCONNECT* with the status set to Established. In raw T1 and raw HDLC modes, the Established message is generated immediately.

By default, the module attempts to establish the D-channel indefinitely until it is successful. The module generates an *L3L4mDISCONNECT* message with a status of Establishing every 4 seconds until the D-channel is established. If the host continues to receive these messages, the application can log the events and cancel the request using an *L4L3mDISABLE_PROTOCOL* message.

In North American Q.931 applications, an *L3L4mB_CHANNEL_STATUS* message might also be received for each B-channel (this is a Central Office or CO option).

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid lapdid value has been used. The only lapdid values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errINVALID_CONN_TYPE 11

Invalid PRI Connection specified.

L3L4errD_CHAN_NOT_DISABLED 12

Indicates the D-channel on the module is already enabled or an attempt was made to mix modes between Raw T1, Raw HDLC, link layer, and so forth. Also issued in response to a IISDNepcmdNEW_HDLC_FLAG_FILL command if the HDLC flag fill feature has not been enabled.

L3L4errINVALID_HDLC_MAPPING 13

Invalid HDLC Bit Mask; channels specified are already in use.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

L3L4errINVALID_INTERFACE 19

For NFAS configurations, indicates value specified is not 0 through 19 or 0xFF.

L3L4errINVALID_LLI 21

For Q.931, indicates the LLI value in the common header is a nonzero value. For all other cases, indicates the LLI value in the common header is an invalid number. Refer to *iisdn.h* for additional information.

L3L4errVC_TABLE_FULL 22

Indicates the maximum number of LLIs has been established for this LAP-D ID stream. The default value is one LLI per physical HDLC data stream.

L3L4errLLI_NOT_FOUND 23

Indicates the LLI was not included in the data structures, or the message contained a IISDNepcmdNEW_HDLC_FLAG_FILL command for a channel that has not been enabled.

L3L4errBLOCKED 24

If overriding the default mezzanine buffer configuration, indicates insufficient memory to allocate internal buffers.

L3L4errNON_NFAS 27

An NFAS function (D-channel switchover) was specified for a non-NFAS configuration.

L3L4errINVALID_STATE 28

A D-channel switchover was requested and the D-channels involved are not in the proper states.

L3L4errDCHAN_TEMP_UNAVAIL 30

A D-channel switchover in progress; applies to NFAS configurations only.

L3L4errTOO_MANY_Q931_STACKS 31

Indicates the maximum number of Q.931 D-channels allowed by the software configuration are already established.

L3L4errDATA_INTERFACE_REQUIRED 33

Indicates that the Layer 2 protocol enabled (such as raw T1 or raw HDLC) requires a Data Interface connection.

L3L4errDATA_INTERFACE_INVALID 34

Indicates the protocol enabled (such as Q.931) does not allow a Data Interface connection.

L3L4errINVALID_BUFFSZ 36

Indicates that the HDLC controller buffer size specified for a channel running raw T1 mode was not evenly divisible by 4.

L3L4errDCHAN_ODD_POINTER_ERROR 42

Host-provided pointers must point to “even” addresses.

L3L4errDCHAN_TOO_FEW_BUFFERS 43

Indicates the message was rejected because it specified less than 3 transmit and receive buffers.

L3L4errDCHAN_TOO_MANY_BUFFERS 44

Indicates that the host attempted to configure more than 255 transmit or receive buffers, the maximum allowed.

L3L4errDCHAN_GIVE_TAKE_NONZERO 45

Indicates the host must set the Give/Take indexes to 0 during module initialization.

L3L4errDCHAN_ZERO_RXBUF_LEN 48

The receive buffers cannot have a length of 0 bytes.

See message details in *L3L4mERROR* on *page 1106*.

Details

The *L4L3mENABLE_PROTOCOL* message IISDN_ENA_PROTO_DATA structure consists of three structures:

- IISDN_LEVEL1_CNFG
  Level 1 configuration parameter: 56K rate adaption.
- IISDN_LEVEL2_CNFG
  Level 2 configuration parameters, including Q.921 (LAP-D).
- IISDN_LEVEL3_CNFG
  Level 3 configuration parameters, including Q.931 ISDN, connected switch type and variant, and additional features.

This message supports all Dialogic® Brooktrout® modules, including those equipped with an optional mezzanine.

The message maps the specified LAP-D ID to an HDLC channel or engine. The HDLC controller circuitry presents up to 64 of these engines (depending on module type and option), with each engine identified by LAP-D ID. Each engine (or LAP-D ID) can be configured to perform packet processing for one or more 64K channels, up to every channel in a 24-channel T1 span. The bit mask *hdlc_channel*s specifies the 64K channel or group of 64K channels to be processed by a specific LAP-D ID.

Note: LAPD-IDs 0 through 31 are always associated with the HDLC circuitry on the Dialogic® Brooktrout® module; LAPD-IDs 32 through 63 are always associated with the HDLC circuitry on the mezzanine.

Dialogic® Brooktrout® firmware supports running the *Q.931 + LAP-D* protocol, which is ISDN Layer 3 call control (D-channel messaging), using an HDLC engine.

Simple ISDN Processing

A simple ISDN scenario, a single *L4L3mENABLE_PROTOCOL* message is used to begin ISDN D-channel message processing. In the case of data calls over ISDN, calls are made in the normal way using the *L4L3mCALL_REQUEST* message. Once the distant end has answered and a talk path has been established, the protocol sends *L4L3mENABLE_PROTOCOL* message for the bearer channel established for the call (indicated by LAP-D ID) to enable the data protocol over the path. To disconnect the call an *L4L3mDISABLE_PROTOCOL* disables the data protocol on the bearer channel. The call is disconnected in the normal way using the *L4L3mCLEAR_REQUEST*.

Using Command Mode

When configured for any Layer 2 protocol, use *command* to perform the following functions on a per-DLCI basis:

- Enable/create a DLCI (default value).
- Establish the data link (send a SABME message).
- Release the data link (send a DISC message).
- Change the number of HDLC flags inserted between frames.

When *command* value specifies either establishing or releasing the data link, all Layer 1, 2, and 3 settings are ignored.

Features that can be configured using this message include:

- *Non-Facility Associated Signaling (NFAS)*
  Enabled by setting the *nfas* bit to 1.
- *B-channel negotiation*
  Set the *b_chan_negot* bit to 1 to enable (additionally, *proc_on_exclusv* alters module processing on incoming calls marked “exclusive”).
- *Channel ID IE processing*
  You can change ISDN Channel ID IE processing by setting *chanid_slot_map*, *sprs_chanid_callproc*, and *no_chanid_callproc*.

Note: NFAS operation is supported for T1 interfaces only.

Determining *switch_type* and *variant* settings

Table 23 indicates the combinations of *switch-type* and *variant* settings supported in the current release of Dialogic® Brooktrout® firmware.

- *Yes* - Dialogic® Brooktrout® modules using the protocol variant have been tested while connected to the specified switch.
- *Yes, not verified* - ISDN is designed to support the combination although testing has not yet been performed.
- *No* - the combination is not currently supported but may be in some later release.
- *Shaded* - the switch does not support the protocol variant.

Table 23. Switch Type & Variant Matrix

Variant	Switch Type								
	AT&T 4ESS	AT&T 5ESS	NT DMS-100	NT DMS-250	MD-110 (T1)	MD-110 (E1)	Siemens EWSD	NTT	Unknown
AT&T Custom	Yes	Yes							
NT Custom			Yes	Yes					
NET-3 NET-5						Yes, not verified	Yes		Yes
1TR6 (PRI) 1TR6 (BRI)						No	No		No
VN3						No	No		No
General ITU-T	No	No	No	No	Yes	Yes	Yes, not verified		Yes
TS014 (PRI)						No	Yes		Yes
JATE (INS-1500)								Yes	No
National ISDN-1		Yes	Yes		Yes, not verified				No
National ISDN-2	Yes, not verified	Yes	Yes	Yes, not verified	Yes, not verified				No

L4L3mFACILITY_REQUEST

Purpose

Causes the Dialogic® Brooktrout® module to transmit a FACILITY message to the network when the module uses Release Link Trunk (RLT) signaling.

Message

IISDN_FACILITY_DATA *facility_data*

Message ID

0x86

Input Fields

```
IISDN_CALL_ID call_id;
unsigned char ie_count;
IISDN_IE_STRUCT ie;
```

Input

call_id

Call ID Structure. See *Call ID (IISDN_CALL_ID)* on *page 854*.

ie_count

IE Count. Specifies the number of Information Elements (IEs) included in the IE Structure in this message.

ie

IE Structure. See *Information Element (IISDN_IE_STRUCT)* on *page 865*.

Output

Return values.

After the module successfully processes this message, it sends a FACILITY message to the network requesting removal of the third party from the call.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid *lapdid* value has been used. The only *lapdid* values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errCALL_REF_ERROR 06

Invalid call reference value specified.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

See message details in *L3L4mERROR* on *page 1106*.

Details

This message is specifically designed to support Release Link Trunk (RLT) signaling on PRI spans connected to DMS-100 and DMS-250 switches. RLT signaling supports *call deflection*, a method for forwarding an incoming call to another destination without allocating any B-channels to the call.

Use *L4L3mUNIVERSAL* for general FACILITY messages.

Note: The Release Link Trunk feature works for DMS-250 switches only, as described in NIS A211-4; no other switch types are currently supported.

The *L4L3mFACILITY_REQUEST* message contains two key structures, *call_id* and the IISDN_IE_STRUCT common structure. When the host determines that an incoming call should be deflected, it must initiate a second call by issuing an *L4L3mCALL_REQUEST* message with *rlt_service* set to 1. This causes the module to send a SETUP message to the network over the D-channel. The network acknowledges this message with either an ALERTING or PROGRESS message containing a Facility IE with a RETURN RESULT component, including the Call ID for the second call (link). An *L3L4mALERTING* message provides this IE to the host, specifically in the IISDN_CALL_ID at the end of the IISDN_AL_CON_DATA structure.

The host must read *call_id* from this structure and use the value to populate the IISDN_CALL_ID structure in an *L4L3mFACILITY_REQUEST* message sent over the first link (using *call_ref* and *L4_ref* of the initial call). The IISDN_IE_STRUCT structure must contain a Facility IE with an INVOKE component indicating that the third party should be removed.

Once the network establishes a direct connection between the originator of the first call and the final destination of the second call, it sends a DISCONNECT message over both links to the module. The host receives an L3L4mDISCONNECT message and an *L3L4mUNIVERSAL* message containing the message ID and IE for a FACILITY message which indicates that the RLT operation is complete.

L4L3mFEATURE_REQUEST

Purpose

Supports ANI on Demand and AT&T Variabill ISDN service features.

Message

IISDN_FEATURE_REQUEST *feature_req*

Message ID

0xB3

Input Fields

unsigned char *feature_type*;
struct *feature*;

IISDN_BILLING_FEATURE *feature*;
unsigned char *billing_change_type*;
unsigned char *hundreds_of_dollars*;
unsigned char *tens_of_dollars*;
unsigned char *dollars*;
unsigned char *tenths_of_dollars*;
unsigned char *hundredths_of_dollars*;

Input

feature_type

Feature Type. Specifies the feature requested by this message.

IISDNftANI_ON_DEMAND 0x01

ANI on Demand being requested for this call.

IISDNftBILLING_CHANGE 0x02

AT&T Variabill billing change being requested for this call.

feature

Feature Data. Contains the information required for the network to process the requested feature. For *feature_type:*

- IISDNftANI_ON_DEMAND, set to 0.
- IISDNftBILLING_CHANGE, use the IISDN_BILLING_FEATURE structure (below).

IISDN_BILLING_FEATURE feature

Billing Structure. Contains billing change and the amount of the change.

feature.billing_change_type

Billing Change Type. Specifies the type of change requested. Possible values include:

IISDNvbfNEW_RATE 0x10

Apply a new rate to this call; rate indicated in the fields that follow.

IISDNvbfFLAT_RATE 0x11

Apply the flat rate charge to this call; rate indicated in the fields that follow.

IISDNvbfPREMIUM_CHARGE 0x12

Apply the premium charge rate to this call; rate indicated in the fields that follow.

IISDNvbfPREMIUM_CREDIT 0x13

Apply the premium credit rate to this call; rate indicated in the fields that follow.

IISDNvbfFREE_CALL 0x18

Apply no billing to this call (free call); populate the fields that follow with ASCII value for zero (0x30).

feature.hundreds_of_dollars

ASCII character for hundreds of dollars. Possible values are 0 to 9 inclusive; specify 0x30 (zero) for *billing_change_type* of IISDNvbfFREE_CALL.

feature.tens_of_dollars

ASCII character for tens of dollars. Possible values are 0 to 9 inclusive; specify 0x30 (zero) for *billing_change_type* of IISDNvbfFREE_CALL.

feature.dollars

ASCII character for dollars. Possible values are 0 to 9 inclusive; specify 0x30 (zero) for *billing_change_type* of IISDNvbfFREE_CALL.

feature.tenths_of_dollars

ASCII character for tenths of dollars. Possible values are 0 to 9 inclusive; specify 0x30 (zero) for *billing_change_type* of IISDNvbfFREE_CALL.

feature.hundredths_of_dollars

ASCII character for hundredths of dollars. Possible values are 0 to 9 inclusive; specify 0x30 (zero) for *billing_change_type* of IISDNvbfFREE_CALL.

Output

Return values.

After the module successfully processes this message, it generates a FACILITY message to the network, requesting the feature change.

- *ANI on Demand feature request* - an *L3L4mANI* message is returned to the host with the status of the request.
- *Variabill feature request* - the *L3L4mBILLING_STATUS* message is used to inform the host of the billing change status.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid lapdid value has been used. The only lapdid values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errCALL_REF_ERROR 06

Invalid call reference value specified.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

L3L4errINVALID_MSG_FOR_STATE 16

Indicates either the Dialogic® Brooktrout® firmware cannot generate a CALL PROCEEDING message for this call because of its current state (CALL PROCEEDING not a valid message for the call state) or the call is in the process of being cleared by the network. In the latter case, this error message is immediately preceded by an *L3L4mCLEAR_REQUEST* or *L3L4mCLEAR_WITH_RESTART_REQUEST* message.

L3L4errSERVICE_NOT_OFFERED 29

Flexible billing is not offered for this call.

See message details in *L3L4mERROR* on *page 1106*.

Details

This message supports the following ISDN service features:

- ANI on Demand – this feature allows the host application to request the ANI (originating number) for an incoming call. The *L3L4mANI* message returns ANI to the host. The Dialogic® Brooktrout® firmware implementation supports ANI on Demand as defined by AT&T.
- AT&T Variabill – this feature allows the receiver of inbound calls to change the billing rate for a call after it has been answered. This feature must be provisioned by the service provider, and its availability for a call is indicated in the incoming SETUP message. This is in turn indicated by the module to the host in *L3L4mSETUP_IND* with the value of *feature_availability*; a value of IISDNfaFLEXIBLE_BILLING means the Variabill feature is available for this call. Billing is changed by the host using *L4L3mFEATURE_REQUEST*. The ASCII characters specify the amount of the change. The status of the billing change request is sent to the application in the *L3L4mBILLING_STATUS* message.

Note: Both features must be ordered and provisioned by the service provider; not all features are available from all service providers.

The IISDN_BILLING_FEATURE structure specifies the type of billing change and the amount of the change.

The ASCII characters zero through nine, inclusive, specify billing rates or hundreds, tens, ones, tenths, and hundredths of dollars. You must populate all dollar fields, even if the value is zero. The meaning of *billing_change_type* is determined by the service provider; contact your carrier for feature availability and specific information.

L4L3mINFO_REQUEST

Purpose

Sends an INFORMATION message in Q.931 applications and/or keypad digits in voice applications.

Message

IISDN_INFO_DATA *info_data*

Message ID

0x8B

Input Fields

```
unsigned char sending_complete;
unsigned char ie_count;
IISDN_CALLED_PARTY called_party;
IISDN_KEYPAD keypad;
unsigned short num_digits;
unsigned char digits[IISDN_MAX_DIGITS+3];
IISDN_CAUSE cause;
IISDN_IE_STRUCT ie;
```

Input

sending_complete

Sending Complete. Specifies inclusion of Sending Complete, indicating that all keypad digits have been sent.

0x00 = not complete

0x01 = Sending Complete

ie_count

IE Count. Specifies the number of Information Elements (IEs) included in the IE Structure in this message.

called_party

Called Party Structure. See *Called Party (IISDN_CALLED_PARTY)* on *page 855*.

keypad

Keypad Structure.

num_digits

Number of Digits. Specifies the number of digits to pass.

digits

Digits. Digit string in ASCII. Number of digits in this string must equal the number specified in *num_digits*. Eight-bit values in this field must be between 0x30 and 0x39 (ASCII 0 through 9).

cause

Cause Data Structure. See *Cause Data (IISDN_CAUSE)* on *page 860.*

ie

IE Structure. See *Information Element (IISDN_IE_STRUCT)* on *page 865*.

Output

Return values.

After the module successfully processes this message, it generates an INFORMATION message to the network, containing the keypad digits to pass to the network. If the message included a Sending Complete indication, the network should respond with a CALL PROCEEDING.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid lapdid value has been used. The only lapdid values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errCALL_REF_ERROR 06

Invalid call reference value specified.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

See message details in *L3L4mERROR* on *page 1106*.

Details

This message usually follows an *L4L3mCALL_REQUEST* that did not contain sufficient called party number information. Typically, this message should be issued in response to a SETUP ACKNOWLEDGE message received from the network (reported in an *L3L4mSETUP_ACK*). The *L4L3mINFO_REQUEST* should contain either a Called Party structure (called party IE) or Keypad structure (keypad facility IE) containing either individual digits or a string of digits.

L4L3mJATE_REDIAL

Purpose

Use *L4L3mJATE_REDIAL* for the following purposes:

- Remove the Redial Restriction on a B-channel when active restriction method is IISDN_JATE_REDIAL_RESTRICTION_15X.
- Specify the maximum number of dialed numbers to be tracked per B-channel form redial restrictions.
- Specify the Japan emergency numbers that are not applied the Jate Redial Restriction.

Message

IISDN_JATE_REDIAL *jate_redial*

Message ID

0xEC

Input Fields

```
IISDNu8bit mode;
IISDNu8bit lapdid;
IISDNu8bit bchannel;
IISDNu8bit max_numbers_tracked;
IISDN_EMERG_DIGITS emergNums[IISDN_MAX_EMERG_NUMS];
```

Input

mode

Values include:

IISDN_JATE_CLEAR_RESTRICTION	1
IISDN_JATE_LOAD_EMERG_NUMS	2
IISDN_JATE_NUMBERS_TRACKED	3

lapdid

Used only when *mode* is IISDN_JATE_CLEAR_RESTRICTION.

For ISDN protocols: Specifies the D-channel where redial restrictions apply.

For CAS protocols: Specifies the trunk/DS1 number where redial restrictions apply.

bchannel

Specifies the B-channel number that has active redial restriction. Used only when *mode* is IISDN_JATE_CLEAR_RESTRICTION. *bchannel* values include:

0 = CAS protocols

1 = ISDN protocols

max_numbers_tracked

Specifies maximum number of dialed numbers per B-channel tracked for redial restrictions. Used with *mode* set to IISDN_JATE_NUMBERS_TRACKED.

emergNums

Array that holds up to 10 emergency numbers for Japan that are not applied redial restrictions.

Output

None

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errINVALID_COMMAND_ARGS

When *mode* is IISDN_JATE_CLEAR_RESTRICTION and active redial restriction method is not IISDN_JATE_REDIAL_15X, or invalid *bchannel* value is specified. This error is also returned when *mode* value is invalid.

L3L4errINVALID_MEM_SIZE

Invalid value for *max_numbers_tracked*: less than 30 or greater than 200.

L3L4errINVALID_MSG_FOR_STATE

When *mode* is IISDN_JATE_NUMBERS_TRACKED and *max_numbers_tracked* has already been specified by a previous *L4L3mJATE_REDIAL*. This error is also returned when *mode* value is invalid.

L4L3mPROGRESS_REQUEST

Purpose

Sends a PROGRESS message to the network, although not required by most ISDN applications.

Message

IISDN_PROGRESS *progress_data*

Message ID

0x82

Input Fields

```
unsigned char coding_standard;
unsigned char location;
unsigned char progress_dscr;
IISDN_CAUSE cause;
IISDN_USER_INFO user_info;
unsigned char ie_count;
IISDN_CALL_ID call_id;
IISDN_IE_STRUCT ie;
```

Input

coding_standard

Coding Standard. Indicates the coding standard used to construct this message. Use the default value of IISDNcodCCITT (0x00), in most cases.

IISDNcodCCITT 0x00

ITU-T coding standard used for this message. Use this value unless the progress indication cannot be represented using standard ITU-T coding.

IISDNcodINTERNATIONAL_STD 0x01

This value is reserved for other International coding standards.

IISDNcodNATIONAL_STD 0x02

National standard coding values not supported by ITU-T coding values used for this message. Recipient of this message should be capable of interpreting this meaning.

IISDNcodSTD_SPF_2_LOC 0x03

Coding standard used is specific to the location receiving the message.

location

Location. Indicates the location of the user for which this message is generated. Possible values include:

IISDNlocUSER 0x00

User.

IISDNlocPVT_LOCAL 0x01

Private network serving the local user.

IISDNlocPUB_LOCAL 0x02

Public network serving the local user.

IISDNlocTRANSIT_NET 0x03

Transit network.

IISDNlocPUB_REMOTE 0x04

Public network serving the remote user.

IISDNlocPVT_REMOTE 0x05

Private network serving the remote user.

IISDNlocINTERNATIONAL 0x07

International network.

IISDNlocBEY_INTERWORK 0x10

Network beyond the interworking point.

progress_dscr

Progress description.

IISDNprogUNKNOWN 0x00

Information not available; default value.

IISDNprogNOT_ISDN_INBAND 0x01

Call is not end-to-end ISDN; additional information for this call may be available in-band. This selection indicates to the destination processor the presence of digits or other in-band signaling to monitor.

IISDNprogDEST_NOT ISDN 0x02

Call destination (called party) is not ISDN.

IISDNprogORIG_NOT_ISDN 0x03

Call origination (calling party) is not ISDN.

IISDNprogRETURNED_ISDN 0x04

Call has been returned to the ISDN.

IISDNprogINBAND_INFO_AVL 0x08

Additional in-band information for this call is available in-band. This selection indicates to the destination processor the presence of digits or other in-band signaling to monitor.

cause

Cause Data Structure. See *Cause Data (IISDN_CAUSE)* on *page 860*.

user_info

User Info Structure. See *User Info (IISDN_USER_INFO)* on *page 875*.

ie_count

IE Count. Specifies the number of Information Elements (IEs) included in the IE Structure in this message.

call_id

Call ID Structure. See *Call ID (IISDN_CALL_ID)* on *page 854*.

ie

IE Structure. See *Information Element (IISDN_IE_STRUCT)* on *page 865*.

Output

Return values.

After the module successfully processes this message, it sends a PROGRESS message to the network.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid *lapdid* value has been used. The only *lapdid* values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

L3L4errLAPDID_NOT_ESTABLISHED 02

ISDN D-channel identified by the LAP-D ID has not been established.

L3L4errCALL_REF_ERROR 06

Invalid call reference value specified.

L3L4errINVALID_B_CHANNEL 07

The LAP-D ID message signifies that the B-channel in the CALL REQ is bad.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

L3L4errINVALID_MSG_FOR_STATE 16

Call state is invalid for processing this message.

L3L4errDCHAN_TEMP_UNAVAIL 30

D-channel switchover in progress; applies to NFAS configurations only.

See message details in *L3L4mERROR* on *page 1106*.

Details

PROGRESS is sent after a SETUP message to provide additional information on the call, including:

- Coding standard used to construct the message.
- Location of the user described by the message.
- Network information, relative to the ISDN.
- User-user information, if required.

This message usually indicates that one of the users involved in the call is not an ISDN user and that additional information might be available in the form of in-band signaling.

If the system where the interface is connected uses the AT&T 4ESS Fast Connect Feature, no PROGRESS or ALERTING messages are required for call connection.

This message supports inclusion of custom Information Elements (IEs) using the IE structure. See *Information Element (IISDN_IE_STRUCT)* on *page 865* for more information.

L4L3mREQ_BOARD_ID

Purpose

Requests the board identification from the module.

Message

IISDN_BOARD_ID *board_id*

Message ID

0xC1

L4L3mREQ_LINE_STATUS

Purpose

Requests the current line status (Layer 1) for the target *lapdid*. Currently this is implemented for the BRI module only.

Message

None.

Message ID

`0xA6`

Input

The L4L3 common header supplies all data for the message.

Output

After the module successfully processes this message, it issues an *L3L4mLINE_STATUS* message.

L4L3mREQ_L2_STATS

Purpose

Requests the Level 2 statistics for the channel specified by the LID in the L4L3 common header.

Message

None.

Message ID

`0xAB`

Input

The L4L3common header supplies all data for the message.

Output

Return values.

After the module successfully processes this message, it issues an *L3L4mL2_STATS* message.

Error Returns

If the module is unable to process the message, the following error codes might return in an *L3L4mERROR* message:

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

The *L3L4mERROR* message is detailed in *L3L4mERROR* on *page 1106*.

Details

The system maintains a set of peg counts for each LID. If more than one virtual circuit is run on a channel, the counts represent an aggregate for all virtual circuits. Following the response to this message each peg count is set to zero.

Peg counts are maintained for the following events:

- IFRAME, RR, RNR, REJ, SABM, SABME, and DISC commands sent or received
- RR, RNR, REJ, DM, UA, and FRMR responses sent or received
- CRC errors received
- Received Queue overruns
- Retransmitted I frames
- Polls where there is no response from the module

L4L3mREQ_PROTOCOL_STATUS

Purpose

Requests the current Layer 2 status for the target D-channel.

Message

None.

Message ID

`0xA5`

Input

The L4L3common header supplies all data for the message.

Output

Return values.

After the module successfully processes this message, it issues an *L3L4mPROTOCOL_STATUS* message.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_OUT_OF_RANGE 01

An invalid *lapdid* value has been used. The only *lapdid* values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).

See message details in *L3L4mERROR* in *page 1106*.

L4L3mRESTART

Purpose

Requests that the connected equipment put a specific channel in an idle state.

Message

IISDN_BCHANNEL_ID *channel*

Message ID

0xA4

Input Fields

```
unsigned long bchannel_mask;
unsigned char bchannel;
unsigned char iface;
unsigned char busy_bit_mask;
unsigned char use_bit_mask;
unsigned long n_bchannel;
IISDN_ROBBED_BIT_DATA robbed_bit_data;
```

Input

bchannel_mask

B-channel Bit Mask. Valid only for multi-rate ISDN calls. Specifies the individual PRI channels used for a multi-rate call. Channels are numbered 1 to 24 (ISDN service in US only) with channel 1 as the least significant bit. First multi-rate channel must also be specified in *bchannel*.

0 = not used

1 = multi-rate call

bchannel

B-channel. Identifies the channel unless *use_bit_mask* is nonzero and NFAS is used. Channels are numbered as listed below:

ISDN 23B+DChannels numbered 1 – 23.

ISDN NFASChannels numbered 1 – 24. The *iface* value indicates the span where the channel resides.

ISDN 384KFirst of six channels used for 384K (H0) call. Possible values are 1, 7, 13 for standard ISDN, or 1, 7, 13, and 19 for ISDN NFAS service.

ISDN 1536KMust be channel 1 for 1536K (H11) call.

multi-rate ISDNFirst channel of multi-rate call (set of channels specified in *bchannel_mask*).

iface

Non-Facilities Associated Signaling (NFAS) Interface. Indicates the span where the channel specified in *bchannel* field resides. The host application must maintain the mapping of span (line) to interface. In this message, *iface* is used only when, in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message, the protocol has done both of the following:

- Enabled B-channel negotiation using *b_chan_negot*
- Enabled NFAS using *nfas*

If either of these features are not enabled, *iface* is ignored and set to 0x00. If B-channel negotiation is enabled and NFAS is not in use, set *iface* to 0xFF.

busy_bit_mask

A value of 1 signifies for the message to go offhook when disabling the B-channel (SW56 only).

use_bit_mask

Use Bit Mask. Specifies *n_bchannel* used to indicate the B-channels for the generated message; allowed for NFAS configurations only.

0x00 = use *bchannel*

0x01 = use *n_bchannel*

n_bchannel

NFAS B-channel. This field is ignored if *cnfg.q931.nfas* is not set in a previous *L4L3mENABLE_PROTOCOL*. Bit mask identifies which B-channels to activate on this interface. T1 B-channels are numbered 1 - 24 (24 is usually reserved for the D-channel). Typical setting is 0x00FFFFFE.

1 = enable a B-channel

0 = disable a B-channel

robbed_bit_data

Unused in this message.

Output

Return values.

ISDN Q.931 Protocol

After the module successfully processes this message, it issues a RESTART message to the network and idles the B-channel. If a call was in progress on this channel, the module generates an *L3L4mCLEAR_WITH_RESTART_REQUEST* message to the host. This is followed by an *L3L4mB_CHANNEL_STATUS* message with *b_channel_status* set to IISDNbcsRESTARTING.

If no call was in progress, the module generates an *L3L4mB_CHANNEL_STATUS* message with *b_channel_status* set to IISDNbcsRESTARTING. When the restart sequence completes, the module generates an *L3L4mB_CHANNEL_STATUS* indicating the channel is in service.

On completion of the restart sequence when *variant* is one of the following, the module sets the service state of the specified B-channel to IS (In Service). An *L3L4mB_CHANNEL_STATUS* message is generated indicating the B-channel is in service:

- IISDNvarNATL_ISDN_1
- IISDNvarNATL_ISDN_2
- ISSDNvarNET5

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errLAPDID_NOT_ESTABLISHED 02

The ISDN D-channel identified by the LAP-D ID is not established.

L3L4errINVALID_B_CHANNEL 07

The LAP-D ID message signifies that the B-channel in the CALL REQ is bad.

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

See message details in *L3L4mERROR* on *page 1106*.

Details

Idling a channel tears down any call associated with that channel. The channel to be idled is specified in the structure IISDN_BCHANNEL_ID.

If in Q.931 ISDN protocol, *variant* equals one of the following, then L4L3RESTART also brings the specified B-channel to In Service (IS):

- `IISDNvarNATL_ISDN_1`
- `IISDNvarNATL_ISDN_2`
- `ISSDNvarNET5`

L4L3mSET_HARDWARE

Purpose

Sets the BRI module-level parameters. Only *line_data[0]* is used.

Message

IISDN_HARDWARE_DATA *hardware_data*

Message ID

0xA7

Input Fields

```
unsigned char bri_l1mode;
unsigned char briL1_cmd;
unsigned char swap_bri_chann;
```

Input

bri_l1mode

BRI Layer 1 Mode. BRI modules only. Specifies the Layer 1 signaling used by the line based on the BRI modules location (customer premise or network). Possible values include:

IISDN_l1mode_Terminal

Place the BRI into TE mode.

IISDN_l1mode_Network

Place the BRI into NT mode. Currently for basic call control test purposes only.

briL1_cmd

BRI Layer1 Command. BRI modules only. Controls the BRI module's Layer 1 interaction with the network. Possible values include:

IISDNcmdL1_NULL

Leave Layer 1 parameters as they are. Use when setting *swap_bri_chann*.

IISDNcmdL1_ACTIVATE

Used to set the BRI mode and start Layer 1.

swap_bri_chann

Swap the B1 and B2 channel on the S/T interface. Set to 1.

L4L3mSETUP_ACK_REQUEST

Purpose

Sends a SETUP ACKNOWLEDGE Q.931 message to the network.

Message

IISDN_SETUP_ACK *setup_ack_data*

Message ID

0x8A

Input Fields

```
unsigned long bchannel_mask;
IISDN_PROGRESS_IND progress_ind;
unsigned char bchannel;
unsigned char iface;
unsigned char ie_count;
IISDN_IE_STRUCT ie;
```

Input

bchannel_mask

B-channel Bit Mask. Valid only for multi-rate ISDN calls. Specifies the individual PRI channels used for a multi-rate call. Channels are numbered 1 to 24 (ISDN service in US only) with channel 1 as the least significant bit. First multi-rate channel must also be specified in *bchannel*.

0 = not used

1 = multi-rate call

progress_ind

Progress Indication Structure. See *Progress Indication (IISDN_PROGRESS)* on *page 867*.

bchannel

B-channel. Identifies the channel to be used for the call. In this message, *bchannel* is used only when B-channel negotiation is enabled by setting *b_chan_negot* in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message.

If this feature is not enabled or an *L4L3mCALL_PROCEEDING_REQUEST* message has already been issued for this call, *bchannel* is ignored and set to 0x00. Channels are numbered as listed below:

ISDN 23B+DChannels numbered 1 – 23.

ISDN NFASChannels numbered 1 – 24. The *iface* value indicates the span where the channel resides.

ISDN 384KFirst of six channels used for 384K (H0) call. Possible values are 1, 7, 13 for standard ISDN, or 1, 7, 13, and 19 for ISDN NFAS service.

ISDN 1536KMust be channel 1 for 1536K (H11) call.

multi-rate ISDNFirst channel of multi-rate call (set of channels specified in *bchannel_mask*).

iface

Non-Facilities Associated Signaling (NFAS) Interface. Indicates the span where the channel specified in *bchannel* field resides. The host application must maintain the mapping of span (line) to interface. In this message, *iface* is used only when, in the IISDN_LEVEL3_CNFG structure of the *L4L3mENABLE_PROTOCOL* message, the protocol has done both of the following:

- Enabled B-channel negotiation using *b_chan_negot*
- Enabled NFAS using *nfas*

If either of these features are not enabled, *iface* is ignored and set to 0x00. If B-channel negotiation is enabled and NFAS is not in use, set *iface* to 0xFF.

ie_count

IE Count. Specifies the number of Information Elements (IEs) included in the IE Structure in this message.

ie

IE Structure. See *Information Element (IISDN_IE_STRUCT)* on *page 865*.

Output

Return values.

After the module successfully processes this message, it constructs the SETUP ACKNOWLEDGE message with the specified IEs and sends it to the network.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errINVALID_COMMAND_ARGS 15

Syntax error; invalid message arguments.

See message details in *L3L4mERROR* on *page 1106*.

Details

This message is typically issued in response to a SETUP message that did not contain sufficient called party information. The SETUP ACKNOWLEDGE indicates to the network that the Dialogic® Brooktrout® module is ready to receive an INFORMATION message that contains a Called Party IE or Keypad facility IE containing either individual digits or a string of digits (this information is reported to the host in an *L3L4mINFO_REQUEST* message).

L4L3mUNIVERSAL

Purpose

Constructs proprietary or other ISDN Q.931/932 messages not supported by Dialogic® Brooktrout® firmware.

Message

IISDN_UNIVERSAL *universal*

Message ID

0x88

Input Fields

```
unsigned char msg_id;
unsigned char ie_count;
unsigned char null_call_ref;
unsigned char q931_escape;
unsigned char aoc[16];
IISDN_IE_STRUCT ie;
```

Input

msg_id

Q.931/932 Message ID. Specifies the type of message being sent. The receiving end should be able to interpret this value.

ie_count

IE Count. Specifies the number of Information Elements (IEs) included in the IE Structure in this message.

null_call_ref

Allows the application to indicate that the call reference field in a generated message must have a NULL value.

q931_escape

To use the Q931 escape message, set *msg_id* to 0 and *q931_escape* to the Q931 escape function code.

aoc

Parsed advice of charge information.

ie

IE Structure. See *Information Element (IISDN_IE_STRUCT)* on *page 865*.

Output

Return values.

Error Returns

If the module is unable to process the message, the protocol might return the following error codes in an *L3L4mERROR* message:

L3L4errNO_CRSTRUCT_AVAILABLE 05

No call record structures are available for this call; maximum number of calls supported by the module already exist.

L3L4errCALL_REF_ERROR 06

Invalid *call_ref* specified in the common header.

L3L4errINVALID_MSG_FOR_STATE 16

Message issued for a call in an idle state.

See message details in *L3L4mERROR* on *page 1106*.

Details

This message allows developers to use proprietary or site-specific messages in an application. In using this message, the following rules apply:

- This message has no effect on call state; do not use for actions that require a change in the call's state machine. Additionally, the application is responsible for ensuring the actions required by this message are consistent with the call's current state.
- Dialogic® Brooktrout® firmware does not check the message ID or message content; this is the responsibility of the application designer. Information Element ordering is also the responsibility of the application designer.
- Messages must observe the longword boundary rules that apply to all BSMI messages; pads have been provided in the message for this purpose.

This message is useful for applications requiring non-call associated signaling. For example, the host can issue *L4L3mUNIVERSAL* with *call_ref* set to 0 (no actual call), a unique *L4_ref* (used by the host to track the message transaction), the *msg_id* for a REGISTER message, and any required IEs to provide information to the network. While call records and call references are designated to this transaction, no B-channels are allocated. The network acknowledges the information with a RELEASE COMPLETE, which is reported to the host in *L3L4mUNIVERSAL*. The host can use *L4_ref* of the *L3L4mUNIVERSAL* to identify the transaction, and delete the call records and references.

The *q931_escape*, in the IISDN_UNIVERSAL structure, is set to 1, the Message ID is prefixed with a 0x00 (a two-byte message).

For example, a HOLD message is 0x00 0x9B (0x00 is the escape). There are supplementary messages for the ATT #5 custom switch. Code the Universal message filling *msg_id* with the message type. Dialogic® Brooktrout® firmware then fills out the escape if value is set to 1. A sample HOLD message follows:

```
L4L3cntlp->data.universal.msg_id = 0x9b; /*HOLD*/
L4L3cntlp->data.universal.q931_escape = 1;
L4L3cntlp->data.universal.ie_count=0;
```

In the L3 to L4 direction, the user knows if the escape message is in the value from the `L4L3cntlp-data.universal.q931_escape` structure. This value equals 1 if escape is contained in this message. The rest of the *L3L4mUNIVERSAL* message remains the same.

Call Hold (ITU-T Q.932, ETSI EN 300 196-1 and ETSI EN 300 141-1) is implemented on the module firmware and accessed through the *L4L3mUNIVERSAL* and *L3L4mUNIVERSAL* messages. Modify the HOLD call state by setting the *universal.msg_id* as follows:

0x24	Request a call be placed on HOLD
0x28	Acknowledge a HOLD request
0x30	Reject a HOLD request (requires addition of a CAUSE IE)
0x31	Request a Call be received
0x33	Acknowledge a RETRIEVE request
0x37	Reject a RETRIEVE request (requires addition of a CAUSE IE)

Add any IE by using the IISDN_IE_STRUCT. If an error occurs when trying to send a HOLD message. The module returns an *L3L4mUNIVERSAL* message with a *univeral.msg_id* of 0x7D (STATUS message) with a CAUSE and CALL STATE IE added. The CAUSE IE is parsed to determine the reason for the rejected message.

The HOLD state machine accepts HOLD requests and passes the request to the host (as a *L3L4mUNIVERSAL* message). The host application can either:

- Accept this message with *L4L3mUNIVERSAL* (*universal.msg_id* = 0x28)
- Reject this message with *L4L3mUNIVERSAL* (*universal.msg_id* = 0x30) and a CAUSE ID added.

The auxiliary HOLD state interacts with the normal call state to determine if a call is disconnected and if so, returns the HOLD state to idle.

L4L3mUSER_INFO

Purpose

Passes up to 130 bytes of information over the network.

Message

IISDN_USER_INFO *user_data*

Message ID

0x87

Input Fields

IISDN_USER_INFO *user_data*;

Input

user_data

User Info Structure. See *User Info (IISDN_USER_INFO)* on *page 875*.

Output

None.

Details

The message consists of the User Info structure documented in *User Info (IISDN_USER_INFO)* on *page 875*.

As a proprietary feature, the user can specify a call reference value of 0x00 rather than the call reference of a valid call. The Dialogic® Brooktrout® module then passes this message to the network using the global call reference.

This message is ignored in call clearing states.

27 - Module to Host (L3L4m) Messages

This chapter explains each of the BSMI L3L4 messages a module sends to the host to provide information about the call.

L3L4 messages are described in alphabetical order beginning on *page 1087*.

L3L4 messages provide the host processor with information on the status of the Dialogic® Brooktrout® module, including module configuration, call status, and events on the network. This interface is supported for the full range of Dialogic® Brooktrout® modules. The C language structures that make up this interface are contained in the iisdn.h include file included in the Dialogic® Brooktrout® firmware release.

Note: In the case of discrepancies between the descriptions in the *iisdn.h* file and this document, follow the structures in *iisdn.h*. Changes in the software since the release of this document are contained in the Release Notes that accompany the product.

L3L4mALERTING

Purpose

Indicates the network received an ALERTING message (the far end is ringing).

Message

IISDN_AL_CON_DATA *al_con_data*

Message ID

0x03

Input Fields

IISDN_AL_CON_DATA *al_con_data;*

Input

al_con_data

Alerting and Connecting Data Message. See *Alerting and Connecting Data Message (IISDN_AL_CON_DATA) on page 851*.

Output

None.

Details

This message is received only if the call is end-to-end ISDN. The structure of this message is identical to that of the following messages:

- *L4L3mALERTING_REQUEST*
- L4L3mCONNECT_REQUEST
- *L3L4mCONNECT*

If the system connected to the interface uses the AT&T 4ESS Fast Connect Feature, no PROGRESS or ALERTING messages are required for call connection.

For the first message received by the host in response to an *L4L3mCALL_REQUEST* message, the application must store the call reference value assigned by the Dialogic® Brooktrout® module. The host requires this value and the L4 reference value for further call control actions by the host.

The IE structure included in this message is unused and therefore, set to zero.

L3L4mANI

Purpose

Provides the ANI (originating number) in response to an *L4L3mFEATURE_REQUEST* message, supporting the ANI on Demand feature.

Message

IISDN_ANI_DATA *ani_data*

Message ID

0x31

Input Fields

```
unsigned char status;
unsigned char filler;
IISDN_CALLING_INFO calling_party;
```

Input

status

ANI Status. Indicates the status of the ANI request. Possible values include:

IISDNaniREJECTED 0x01

ANI request was rejected because ANI was not available, the feature requested is not provisioned (subscribed), or the feature has not been implemented. All other fields in this message are zeroed out.

IISDNaniACCEPTED 0x02

ANI request was accepted. ANI is contained in the Calling Party structure at the end of the message.

IISDNaniERROR 0x03

ANI request contained an error. Application checks original *L4L3mFEATURE_REQUEST* message and resends. All other fields in this message are zeroed out.

filler

Filler byte reserved for future use; set to 0x00.

calling_party

Calling Party Structure. See *Calling Party (IISDN_CALLING_PARTY) on page 857*.

Output

None.

Details

This ANI on Demand feature allows the host application to request the ANI (originating number) for an incoming call. ANI is supplied using *Calling Party (IISDN_CALLING_PARTY) on page 857*. This implementation supports ANI on Demand as defined by AT&T.

L3L4mB_CHANNEL_STATUS

Purpose

Informs the host of the status of the B-channel, issued in response to a change of status in the B-channel.

Message

None.

Message ID

`0x22`

Input Fields

`unsigned char` *`b_channel_status`*;

Input

`b_channel_status`

B-channel Status. Indicates the status at the time of the message. Possible values include:

IISDNbcsOUT_OF_SERVICE 0x00

B-channel is out of service.

IISDNbcsRESTARTING 0x01

B-channel is restarting.

IISDNbcsMAINTENANCE 0x02

B-channel is in maintenance mode.

IISDNbcsIN_SERVICE 0x03

B-channel is in service.

Output

No response is expected for out of service or in service messages. Dialogic® Brooktrout® firmware supports a far-end maintenance mode only.

When the module sends this message with a value of IISDNbcsMAINTENANCE, the network (Central Office) expects to accept test calls on this B-channel. It also sets the B-channel to loopback using the appropriate switching facility function call define in the Bfv API. When the module completes network testing, the

B-channel is returned to service and the host receives this message with a value of IISDNbcsIN_SERVICE. At this point, remove any loopback connection.

Details

For ISDN, the module sends this message after receiving of a SERVICE or SERVICE ACK message from the network.

This message also indicates a B-channel restart condition. If the module restarts the channel, a value of IISDNbcsRESTARTING indicates the change of state. When the module restores the B-channel to service, it sends a second *L3L4mB_CHANNEL_STATUS* message with a value of IISDNbcsIN_SERVICE. You cannot use the B-channel in a call until the in-service indication is received.

L3L4mBILLING_STATUS

Purpose

Provides the status of the billing change request in response to an *L4L3mFEATURE_REQUEST* message, supporting the Variabill feature.

Message

None.

Message ID

`0x30`

Input Fields

`unsigned short` *`billing_change_status`*;

Input

`billing_change_status`

Billing Change Status. Indicates the status of the Variabill feature billing change request. Possible values include:

IISDNvbsBCHG_ACCEPTED 0x01

Billing change request was accepted and change made.

IISDNvbsBCHG_DENIED 0x02

Billing change request was denied either because the call is using a service that does not support flexible billing or the change was denied by the network. No change to billing made.

IISDNvbsBCHG_REJECTED 0x03

Billing change request was rejected due to a badly formatted message or some type of protocol error. A timer is started by the network to wait for a correct message.

Output

None.

Details

This feature allows the host application to request a change in the billing rate for an incoming call. This implementation supports Variabill as defined by AT&T for the 4ESS switch.

L3L4mBOARD_ID

Purpose

Sends the board identification to the host.

Message

IISDN_BOARD_ID *board_id*

Message ID

0x3B

Input Fields

```
char iisdn_ver[32];
char banner[32];
char date[16];
char model[16];
char rev;
unsigned char board_type;
unsigned char num_hdlc_chan;
unsigned char num_modem_chan;
unsigned char num_lines;
unsigned char line_type[IISDN_MAX_LINES];
unsigned char kernal_ram_size;
unsigned char num_bfio_devices;
unsigned char devType;
unsigned char firstLapdid;
unsigned char numLapdids;
```

Input

iisdn_ver[32]

The current version of the ISDN stack.

banner[32]

Software copyright information.

date[16]

Date in format: Mm dd yyyy

model[16]

Model name of the module.

rev

Current revision of the module.

board_type

FF

num_hdlc_chan

Number of HDLC channels supported by the module.

num_modem_chan

Number of modem channels supported by the module.

num_lines

Number of trunks or ports the module supports. Identifies the configuration (T1, E1 or BRI) for each line on the module. Possible values include:

IISDNline_typeUNKNOWN

Could not identify hardware configuration.

IISDNline_type PRI_T1

PRI T1 trunk, no on-board CSU present.

IISDNline_type PRI_T1CSU

PRI T1 trunk, optional on-board CSU.

IISDNline_typePRI_E1

PRI E1 trunk.

line_type[IISDN_MAX_LINES]

1 = T1

3 = E1

5 = BRI

kernel_ram_size

Ram size of the kernel.

num_bfio_devices

Number of buffered I/O devices. These currently include HDLC controllers associated with framer chips.

devType

Device type.

firstLapdid

Default ID for the first channel.

numLapdids

Number of channels.

L3L4mCALL_PROCEEDING

Purpose

Indicates receipt of a call proceeding message. Use *L4L3mENABLE_PROTOCOL* to enable *L3L4mCALL_PROCEEDING*.

Message

IISDN_CALL_PROC_DATA *report_incoming_callproc*

Message ID

0x3A

Input Fields

IISDN_PROGRESS_IND *progress_ind*;

Input

progress_ind

Progress Indication Structure. See *Progress Indication (IISDN_PROGRESS) on page 867*.

L3L4mCALL_PROC_SENT

Purpose

Indicates that the protocol sent a call message through the network.

Message

None.

Message ID

`0x39`

Details

To enable this feature, the user must set *send_l3l4_callproc* in the *L4L3mENABLE_PROTOCOL* message that configured the specific *lapdid*.

L3L4mCLEAR_REQUEST

Purpose

Acknowledges receipt of an *L4L3mCLEAR_REQUEST* message or an unsolicited notification indicates that a call has cleared.

Message

IISDN_CLR_DATA *clr_data*

Message ID

0x06

Input Fields

```
IISDN_CAUSE Boston;
IISDN_USER_INFO user_info;
IISDN_PROGRESS_IND progress_ind;
unsigned char ie_count;
IISDN_CONNECTED_ADDRESS connected_address;
IISDN_IE_STRUCT ie;
```

Input

cause

Cause Data Structure. See *Cause Data (IISDN_CAUSE) on page 860*.

user_info

User Info Structure. See *User Info (IISDN_USER_INFO) on page 875*.

progress_ind

Progress Indication Structure. See *Progress Indication (IISDN_PROGRESS) on page 867*.

ie_count

IE Count. Number of Information Elements (IEs) included in the IE Structure in this message.

connected_address

Connected Address Structure. See *Cause Data (IISDN_CAUSE) on page 860*.

ie

IE Structure. Unused in this message.

Output

None.

Details

The network sent a RELEASE or RELEASE COMPLETE message. The protocol cleared the call and the B-channel is idle. The structure of this message is identical to that of the following messages:

- *L4L3mCLEAR_REQUEST*
- *L3L4mDISCONNECT*

The Dialogic® Brooktrout® module *always* generates either an *L4L3mCLEAR_REQUEST* or *L3L4mCLEAR_WITH_RESTART_REQUEST* when call clearing is completed as directed by an *L4L3mCLEAR_REQUEST*. An *L3L4mDISCONNECT* indicating *all_calls_dropped* also indicates a call is cleared. The B-channel cannot be used for a new call until one of these messages is received by the host.

Although included in this message, the IE structure is unused in messages from the module to the host and it is set to zero.

L3L4mCLEAR_WITH_RESTART_REQUEST

Purpose

Indicates that the protocol cleared the call due to a network detected error.

Message

IISDN_CLR_DATA *clr_data*

Message ID

0x08

Input Fields

```
IISDN_CAUSE cause;
IISDN_USER_INFO user_info;
IISDN_PROGRESS_IND progress_ind;
unsigned char ie_count;
IISDN_CONNECTED_ADDRESS connected_address;
IISDN_IE_STRUCT ie;
```

Input

cause

Cause Data Structure. See *Cause Data (IISDN_CAUSE) on page 860*.

user_info

User Info Structure. See *User Info (IISDN_USER_INFO) on page 875*.

progress_ind

Progress Indication Structure. See *Progress Indication (IISDN_PROGRESS) on page 867*.

ie_count

IE Count. Number of Information Elements (IEs) included in the IE Structure in this message.

connected_address

Connected Address Structure. See *Cause Data (IISDN_CAUSE) on page 860*.

ie

IE Structure. Unused in this message.

Output

None.

Details

The Dialogic® Brooktrout® module *always* generates either an *L3L4mCLEAR_REQUEST* or *L3L4mCLEAR_WITH_RESTART_REQUEST* message when the protocol clears a call as directed by an *L4L3mCLEAR_REQUEST* message. A call is also cleared when an *L3L4mDISCONNECT* message contains *all_calls_dropped*. You cannot use the B-channel for a new call until the host receives one of these messages.

When the protocol restarts a B-channel, the host issues an *L3L4mDISCONNECT* message with *b_channel_status* equal to IISDNbcsRESTARTING. A subsequent *L3L4mDISCONNECT* message indicates that the restarted B-channel is back in service.

If the network sent the RESTART, the protocol restarts the B-channel but does not generate an *L3L4mDISCONNECT* message. The module responds to the network with a RESTART ACK message and clears the call, if any. After the restart process competes, the host generates an *L3L4mCLEAR_WITH_RESTART_REQUEST* message.

L3L4mCONNECT

Purpose

Indicates a CONNECT message received from the network. The far end answered and a stable call is established.

Message

IISDN_AL_CON_DATA *al_con_data*

Message ID

0x04

Input Fields

IISDN_AL_CON_DATA *al_con_data;*

Input

al_con_data

Alerting and Connecting Data Message. See *Alerting and Connecting Data Message (IISDN_AL_CON_DATA) on page 851*.

Output

None.

Details

This message structure is identical to that of the following messages:

- *L4L3mALERTING_REQUEST*
- *L4L3mCONNECT_REQUEST*
- *L3L4mALERTING*

If the system connected to the interface uses the AT&T 4ESS Fast Connect Feature, no PROGRESS or ALERTING messages are required for call connection. In this case, this is the first message from the Dialogic® Brooktrout® module in response to an *L4L3mCALL_REQUEST*. Therefore, the application must store the call reference value assigned by the module. The host requires this value and the L4 reference value for further call control actions.

Although included in this message, the IE structure is unused in messages from the module to the host and it is set to zero.

L3L4mCONN_ACK_IND

Purpose

Indicates that a *connect_ack* message is received by the controller.

Message

IISDN_Q931_CNFG *q931*

Message ID

0x0C

Details

This is an optional message, and only received if enabled by *subscribe_connack* in the *L4L3mENABLE_PROTOCOL* message.

L3L4mDISCONNECT

Purpose

Indicates receipt of a DISCONNECT message from the network.

Message

IISDN_CLR_DATA *clr_data*

Message ID

0x05

Input Fields

```
IISDN_CAUSE cause;
IISDN_USER_INFO user_info;
IISDN_PROGRESS_IND progress_ind;
unsigned char ie_count;
IISDN_CONNECTED_ADDRESS connected_address;
IISDN_IE_STRUCT ie;
```

Input

cause

Cause Data Structure. See *Cause Data (IISDN_CAUSE) on page 860*.

user_info

User Info Structure. See *User Info (IISDN_USER_INFO) on page 875*.

progress_ind

Progress Indication Structure. See *Progress Indication (IISDN_PROGRESS) on page 867*.

ie_count

IE Count. Number of Information Elements (IEs) included in the IE Structure in this message.

connected_address

Connected Address Structure. See *Cause Data (IISDN_CAUSE) on page 860*.

ie

IE Structure. Unused in this message.

Output

The host must clear the call by issuing an *L4L3mCLEAR_REQUEST* message.

The module must receive an *L3L4mCLEAR_REQUEST* message before you can use the B-channel for another call.

Details

The far end has disconnected and clears the call. The structure of this message is identical to the following messages:

- *L4L3mCLEAR_REQUEST*
- *L3L4mCLEAR_REQUEST*

Although included in this message, the IE structure is unused in messages from the Dialogic® Brooktrout® module to the host and it is set to zero.

L3L4mERROR

Purpose

Indicates either the host specified invalid information in an L4L3 message or an error condition occurred for the specified interface.

Message

None.

Message ID

`0x20`

Input Fields

`unsigned char` *`error_code`*;

Input

`error_code`

Error Code. Specifies the cause of the *L3L4mERROR* message. Possible values are:

Error and Meaning	Value
L3L4errNO_ERROR	00
No error; internal use only.	
L3L4errLAPDID_OUT_OF_RANGE	01
An invalid *lapdid* value has been used. The only *lapdid* values allowed are 0 (span 1), 2 (span 2), 4 (span 3) or 6 (span 4).	
L3L4errLAPDID_NOT_ESTABLISHED	02
LAPD ID processing is not established or the link layer protocol did not establish or was dropped. Use the *L4L3mENABLE_PROTOCOL* message to establish LAPD processing, then resend the message.	
L3L4errINVALID_CALLED_NUMBER	03
Invalid Called Number specified in an ISDN Call Control message. Called Number was longer than 24 digits for all calls. Resends the message with the correct Called Number.	
L3L4errNO_CRV_AVAILABLE	04
No more call reference numbers are available.	
L3L4errNO_CRSTRUCT_AVAILABLE	05
No call record structures are available for allocation. Indicates the call cannot be processed at this time because the module exhausted all internal resources.	

Error and Meaning	Value
L3L4errCALL_REF_ERROR	06
Invalid call reference specified; Dialogic® Brooktrout® firmware could not locate the call record by either *call_ref* or *L4_ref* values. This error might indicate the race condition that occurs when both the Dialogic® Brooktrout® module and host send Clear Request messages. No further action is required.	
L3L4errINVALID_B_CHANNEL	07
Invalid B-channel specified in *L4L3mCALL_REQUEST*. Resend the message with a valid B-channel port number specified.	
L3L4errB_CHANNEL_RESTARTING	08
The module restarts the B-channel specified in response to a network error. The module clears the call on that B-channel. The B-channel is available for another call once an *L3L4mB_CHANNEL_STATUS* message indicates the channel is in service received.	
L3L4errB_CHANNEL_OOS	09
B-channel specified is out of service. Resend the message with a different B-channel specified.	
L3L4errINVALID_CALL_TYPE	10
An outgoing call cannot be placed on this interface; *call_type* is most likely invalid.	
L3L4errINVALID_CONN_TYPE	11
L4L3mENABLE_PROTOCOL message specified a connection type as invalid. Resend the message with a valid connection type.	
L3L4errD_CHAN_NOT_DISABLED	12
The module received an *L4L3mENABLE_PROTOCOL* message for a D-channel that was already enabled.	
L3L4errINVALID_HDLC_MAPPING	13
L4L3mENABLE_PROTOCOL message specifies the HDLC bit mask as invalid; some of the specified channels are in use. Resend the message with a valid HDLC bit mask.	
L3L4errINVALID_DATA_QUEUE	14
The host setup invalid data queue buffers for communication with the module. Give/Take ≠0 or the number of TX/RX buffers defined is less than two.	
L3L4errINVALID_COMMAND_ARGS	15
Syntax error in the received message. Correct and resend the message.	
L3L4errINVALID_MSG_FOR_STATE	16
Control message received is invalid for the current state of the specified call.	
L3L4errDATA_PACKET_LOST	17
Data packets lost because the TX buffer length is too big to fit into the internal buffer.	
L3L4errPM_NOT_ESF	18
Performance monitoring cannot be enabled on a non-ESF link.	

Error and Meaning	Value
L3L4errINVALID_INTERFACE	19
NFAS. Invalid interface specified.	
L3L4errB_CHANNEL_INUSE	20
Rejected Call Request.	
L3L4errINVALID_LLI	21
Invalid LLI number. Refer to the description of LLI in *Logical Link ID or DLCI on page 849*.	
L3L4errVC_TABLE_FULL	22
Maximum of eight LLIs per physical channel.	
L3L4errLLI_NOT_FOUND	23
LLI was not in the data structures.	
L3L4errNO_HARDWARE	25
The hardware required for the request is not present on this module type.	
L3L4errNON_NFAS	27
Indicates the message was rejected because the channel specified is not part of an NFAS group.	
L3L4merrINVALID_STATE	28
The channel or call specified in the message is not in the correct state to process this message.	
L3L4merrSERVICE_NOT_OFFERED	29
Indicates the network rejected the message because the network does not support the feature (ANI on Demand, Variabill) requested.	
L3L4errTOO_MANY_Q931_STACKS	31
Indicates the module rejected the message because the maximum number of Q.931 D-channels allowed are already established.	
L3L4errDATA_INTERFACE_REQUIRED	33
The Layer 2 protocol enabled (such as raw T1 or raw HDLC) requires a Data Interface connection.	
L3L4errDATA_INTERFACE_INVALID	34
The protocol enabled (such as Q.931) does not allow a Data Interface connection.	
L3L4errINVALID_BUFFSZ	36
The HDLC controller buffer size specified for a channel running raw T1 mode was not evenly divisible by 4.	
L3L4errDCHAN_ODD_POINTER_ERROR	42
Indicates host-provided pointers must point to “even” addresses.	

Error and Meaning	Value
L3L4errDCHAN_TOO_FEW_BUFFERS	43
Indicates the module rejected the message because it specified less than 3 transmit and receive buffers.	
L3L4errDCHAN_TOO_MANY_BUFFERS	44
Indicates that the host attempted to configure more than 255 transmit or receive buffers.	
L3L4errDCHAN_GIVE_TAKE_NONZERO	45
Indicates the host must set the Give/Take indexes to 0 during module initialization.	
L3L4errDCHAN_ZERO_RXBUF_LEN	46
Indicates that the receive buffers cannot have a length of 0 bytes.	
L3L4errDLCI_MANDITORY	48
Indicates the module rejected the message because the L3L4 common header did not specify a DLCI. A DLCI is required for LAP-D data connections.	
L3L4errCHAN_KBIT_RATE_BAD	49
Indicates the Level 1 parameter channel kilobit rate specified in the *L4L3mENABLE_PROTOCOL* message is outside the valid range of 0 to 4096 kilobits.	
L3L4errTX_BUFFER_MISALIGNED	52
Indicates that transmit buffers on the host side of the data interface must be aligned on 256 K boundaries.	
L3L4errRX_BUFFER_MISALIGNED	53
Indicates that receive buffers on the host side of the data interface must be aligned on 256 K boundaries.	
L3L4errTOO_MANY_DLCIS	54
Indicates the number of DLCIs specified for a particular channel exceeded the maximum (set by IISDN_MAX_VC_PER_CHAN).	
L3L4errINVALID_SMI_MSGID	59
BSMI message type is invalid.	
L3L4errINVALID_CLOCK_MODE	60
Invalid clocking type.	
L3L4errNO_OVERFLOW_QUEUE	61
Indicates initialization of the overflow queue failed. The queue must be initialized for the overflow queue to function.	
L3L4errSTATUS_IGNORED	100
Ignored a status message from AT&T Maintenance processing. *L4_ref* set to 0xFFFF.	
L3L4errBAD_CALL_REF	101
The network sends the call reference number because of an invalid maintenance message. *L4_ref* set to 0xFFFF.	

Output

None.

Details

When the protocol generates the message in response to a host message, use the L4 Reference value assigned by the host. If the protocol generates the message due to an autonomous event in the network, the L4 Reference value is set to 0xFFFF.

L3L4mINFO_REQUEST

Purpose

The network sends an INFORMATION message (in Q.931 applications) or a keypad digit string.

Message

IISDN_INFO_DATA *info_data*

Message ID

0x8B

Input Fields

```
unsigned char sending_complete;
unsigned char ie_count;
IISDN_CALLED_PARTY called_party;
IISDN_KEYPAD keypad;
unsigned short num_digits;
unsigned char digits[IISDN_MAX_DIGITS];
IISDN_CAUSE cause;
IISDN_IE_STRUCT ie;
```

Input

sending_complete

Sending Complete. Indicates that the network transmitted all information, including all keypad digits.

ie_count

IE Count. Number of Information Elements (IEs) included in the IE Structure in this message.

called_party

Called Party Structure. See *Called Party (IISDN_CALLED_PARTY) on page 855*.

keypad

Keypad Structure. See *num_digits*, *digits*.

num_digits

Number of Digits. Specifies the number of digits received.

digits[IISDN_MAX_DIGITS]

Digits. Digit string in ASCII. Number of digits in this string should equal the number specified in *num_digits*. Eight-bit values in this field must be between 0x30 and 0x39 (ASCII 0 through 9).

cause

Cause Data Structure. See *Cause Data (IISDN_CAUSE) on page 860*.

ie

IE Structure. Unused in this message.

Output

The host issues an *L4L3mCALL_PROCEEDING_REQUEST* to continue the call.

Details

This message usually appears in incoming call scenarios when the network sends a SETUP message that did not contain sufficient called party information. Typically, this message is in response to an *L4L3mSETUP_ACK_REQUEST*. This message contains either a Called Party structure (called party IE) or Keypad structure (Keypad facility IE) containing either individual digits or a string of digits.

L3L4mL2_STATS

Purpose

Provides Level 2 statistics for the HDLC channel indicated in the *L4L3mREQ_L2_STATS* where it is generated.

Message

IISDN_L2_STATS *stat_data*

Message ID

0x25

Input

frame_tx

Count of IFRAME commands sent and successfully acknowledged.

rr_cmd_tx

Count of RR commands sent.

rnr_cmd_tx

Count of RNR commands sent.

rej_cmd_tx

Count of REJ commands sent.

sabm_tx

Count of SABM commands sent.

sabme_tx

Count of SABME commands sent.

disc_tx

Count of DISC commands sent.

rr_rsp_tx

Count of RR responses sent.

rnr_rsp_tx

Count of RNR responses sent.

rej_rsp_tx

Count of REJ responses sent.

dm_tx

Count of DM responses sent.

ua_tx

Count of US responses sent.

frmr_tx

Count of FRMR responses sent.

iframe_rx

Count of IFRAME commands received.

rr_cmd_rx

Count of RR commands received.

rnr_cmd_rx

Count of RNR commands received.

rej_cmd_rx

Count of REJ commands received.

sabm_rx

Count of SABM commands received.

sabme_rx

Count of SABME commands received.

disc_rx

Count of DISC commands received.

rr_rsp_rx

Count of RR responses sent.

rnr_rsp_rx

Count of RNR responses sent.

rej_rsp_rx

Count of REJ responses received.

dm_rx

Count of DM responses received.

ua_rx

Count of REJ responses received.

frmr_rx

Count of FRMR responses received.

crc_errors

Count of CRC errors received.

rcv_errors

Received. Queue over runs.

retrans_cnt

Count of retransmitted Information Frames.

poll_errors

Count of polls without a response.

ui_tx

Count of UI frames transmitted.

ui_rx

Count of UI frames received.

Details

The system maintains a set of peg counts for each Logical ID (LID). If more than one virtual circuit is run on a channel, the counts represent an aggregate for all virtual circuits. Counts are reset to zero following generation of this message.

L3L4mLINE_STATUS

Purpose

Indicates the InstantISDN generated response to the *L4L3mREQ_LINE_STATUS* message.

Message

IISDN_LINE_STATUS *line_status*

Message ID

0x23

Input Fields

IISDN_LINE_STATUS *IISDN_ALARM_STATUS.rcv_red*;
IISDN_LINE_STATUS *l1_state[IISDN_MAX_LINES]*;
IISDN_LINE_STATUS *line_type[IISDN_MAX_LINES]*;

Input

IISDN_ALARM_STATUS.rcv_red

Set to 1, if line is synchronized.

l1_state[IISDN_MAX_LINES]

Layer 1 states are defined in the ITU I.430 standard for BRI.

Value	TE Interface	NT Interface
1	Inactive	Deactivated
2	Sensing	Pending activation
3	Deactivated	Activated
4	Awaiting signal	Pending deactivation
5	Identifying signal	
6	Synchronized	
7	Activated	
8	Lost Framing	

line_type[IISDN_MAX_LINES]

This field reports the current type of line associated with each index in the array. Normally all lines on a Dialogic® Brooktrout® module are the same type.

IISDNline_typeUNKNOWN	Unpopulated network interface
IISDNline_typePRI_T1	North American DSX-1 interface
IISDNline_typePRI_T1CSU	North American DS-1 “long haul” interface
IISDNline_typePRI_E1	2.048 CEPT DS1 interface
IISDNline_typeBRI_U	North American 2-wire Basic Rate ISDN Interface
IISDNline_typeBRI_ST	4-wire Basic Rate ISDN interface

Details

Only the BRI interface supports this message. The protocol reports only one *lapdid* at a time (the one associated with the *L4L3mREQ_LINE_STATUS* message).

The *Input* paragraph for this message only describes the fields applicable to *L3L4mLINE_STATUS*. Any field present in the *L3_to_L4_struct* but not described cannot contain useful information.

L3L4mPROGRESS

Purpose

Indicates a PROGRESS message received from the network (an outgoing call is in process).

Message

IISDN_PROGRESS *progress_data*

Message ID

0x02

Input Fields

IISDN_PROGRESS *progress_data*;

Input

progress_data

Progress Indication Structure. See *Progress Indication (IISDN_PROGRESS) on page 867*.

Output

None.

Details

Indicates the call is not end-to-end ISDN. If the system connected to the interface uses the AT&T 4ESS Fast Connect Feature, no PROGRESS or ALERTING messages are required for call connection.

When this is the first message received by the host in response to an *L4L3mCALL_REQUEST* message, the application must store the *call_ref* assigned by the Dialogic® Brooktrout® module. The host requires *call_ref* and the *L4_ref* for further call control actions.

Although included in this message, the IE structure is unused in messages from the module to the host and it is set to zero.

L3L4mPROTOCOL_STATUS

Purpose

Informs the host of the status of the D-channel.

Message

IISDN_D_CHAN_STAT *d_chan_data*

Message ID

0x21

Input Fields

```
unsigned char status;
unsigned char l2_state;
unsigned char l2_error;
unsigned char l2_errpt;
unsigned long b_channels;
unsigned long b_chan_req;
unsigned long txcount;
unsigned long rxcount;
unsigned short l2_detail;
unsigned char all_calls_dropped;
unsigned long n_b_channels[IISDN_NUM_DS1_INTERFACES];
unsigned long n_b_chan_req[IISDN_NUM_DS1_INTERFACES];
unsigned char nfas_primary_dchan_status;
```

Input

status

Current D-Channel Status. Indicates the status at the time of the message. Possible values include:

IISDNdsNOT_ESTABLISHED 0x00

D-channel is not established.

IISDNdsESTABLISHING 0x01

D-channel is being established.

IISDNdsESTABLISHED 0x02

D-channel is established.

l2_state

Layer 2 State. Indicates the Layer 2 status of the D-channel. Possible values include:

IISDNlpdsTEI_UNASSIGNED 0x00

Dialogic® Brooktrout® module is waiting for the network to assign a Terminal Endpoint Identifier (TEI).

IISDNlpdsTEI_ASSIGNED 0x04

Link in TEI-assigned state.

IISDNlpdsAWAITING_ESTABLISHMENT 0x05

Link in awaiting-establish state.

IISDNlpdsAWAITING_RELEASE 0x06

Link in awaiting-release state; DISC command received and channel release process started.

IISDNlpdsMULTIFRAME_ESTABLISHED 0x07

Link in multiple-frame-established state.

IISDNlpdsTIMER_RECOVERY 0x08

Link in timer recovery state; T200 timer expired and retransmission counter reset.

l2_error

Layer 2 MDL Error Code. Indicates the MDL-ERROR message type reported by the network. Possible values include:

IISDNl2errNO_ERROR 0x00

No MDL-ERROR reported.

IISDNl2errA 0x01

Error Code A. The network received an unsolicited supervisory message; F = 1.

IISDNl2errB 0x02

Error Code B. The network received an unsolicited DM message; F = 1.

IISDNl2errC 0x03

Error Code C. The network received an unsolicited UA message; F = 1.

IISDNl2errD 0x04

Error Code D. The network received an unsolicited UA message; F = 0.

IISDNl2errE 0x05

Error Code E. The network received an unsolicited DM message; F = 0.

IISDNl2errF 0x06

Error Code F. The network received a SABME message to re-establish the connection.

IISDNl2errG 0x07

Error Code G. Unsuccessful retransmission of a SABME message (N200 counter exceeded).

IISDNl2errH 0x08

Error Code H. Unsuccessful retransmission of a DISC message (N200 counter exceeded).

IISDNl2errI 0x09

Error Code I. Unsuccessful retransmission of a STATUS ENQUIRY (N200 counter exceeded).

IISDNl2errJ 0x0A

Error Code J. N (R) error occurred.

IISDNl2errK 0x0B

Error Code K. The network received a FRMR response.

IISDNl2errL 0x0C

Error Code L. The network received a non-implemented frame.

IISDNl2errM 0x0D

Error Code M. The network received an I frame that is not permitted.

IISDNl2errN 0x0E

Error Code N. The network received an I frame of the wrong size.

IISDNl2errO 0x0F

Error Code O. The network received an I frame with too many octets (N201 exceeded).

`l2_errpt`

Layer 2 Error Pointer.

b_channels

In-Service B-channels. Bit mask that unidentified B-channels currently in service on this interface. B-channels are numbered 1 - 24. When configured for NFAS, value is 0x00000000. Possible values for each bit position include:

B-channel is not active.	0x00000000
B-channel is in service.	0x00000001

b_chan_req

Request for Active B-channels. Bit mask that identifies B-channels the host requested in service using the *L4L3mENABLE_PROTOCOL* or *L4L3mENABLE_B_CHANNEL*. B-channels are numbered 1 - 23; 24 is reserved for the D-channel. When configured for NFAS, value is 0x00000000. Possible values for each bit position include:

B-channel is not active.	0x00000000
B-channel is active.	0x00000001

txcount

TX Count. Number of Level 2 packets transmitted by the Dialogic® Brooktrout® module.

rxcount

RX Count. Number of Level 2 packets received by the Dialogic® Brooktrout® module.

l2_detail

Level 2 Detail. Link Layer 2 information for this interface. The protocol enables Level 2 detail including the connection type IISDNctL2_DETAIL.

Values marked with an asterisk (*) can be received regardless of whether L2_DETAIL has been set. Possible values include:

IISDNdsmskDM_RCVD	0x0001
DM received with F bit set.	
IISDNdsmskSABME_RCVD*	0x0002
SABME/SABM received.	

IISDNdsmskSABME_SENT 0x0004

SABME/SABM sent.

IISDNdsmskFRAME_MOD_8 0x0008

Received frame with non-integral octets.

IISDNdsmskBAD_CRC 0x0010

Received frame with bad CRC.

IISDNdsmskBAD_LEN 0x0020

Received frame with bad length.

IISDNdsmskUNKN_CTRL 0x0040

Received frame with unknown control field.

IISDNdsmskUNKN_DLCI 0x0080

Received frame with unknown address.

IISDNdsmskUNEXPECTED 0x0100

Received valid message type in bad state.

IISDNdsmskDISC_RCVD* 0x0200

Received disconnect message.

IISDNdsmskT200* 0x0400

T200/N200 timeout occurred.

IISDNdsmskUA_RCVD 0x1000

UA received.

all_calls_dropped

All Calls Dropped. Used with NFAS configurations. Indicates either the network sent a RESTART message specifying to drop all calls associated with this D-channel, or a D-channel switchover failed. This message replaces the *L3L4mCLEAR_WITH_RESTART_REQUEST* message in the RESTART scenario only. Possible values include:

IISDNacdNO_CHANGE 0x01

No calls were dropped.

IISDNacdRESTART 0x02

Restart message received; all calls dropped.

IISDNacdDCHAN_SWITCHING 0x03

D-channels switch over is in progress; all non-stable calls dropped (NFAS configurations with D-channel backup only).

IISDNacdDCHAN_FAULT 0x04

Backup D-channel did not come into service; all calls dropped (NFAS configurations with D-channel backup only).

n_b_channels[IISDN_NUM_DS1_INTERFACES]

In-Service B-channels. For NFAS configurations only, an array of [IISDN_NUM_DS1_INTERFACES] bit masks that identifies the B-channels currently in service on this interface. B-channels are numbered 1 - 24. For non-NFAS configurations, this value is set to 0x00000000. Possible values for each bit position include:

B-channel is not active. 0x00000000

B-channel is active. 0x00000001

n_b_chan_req[IISDN_NUM_DS1_INTERFACES]

Request for Active B-channels. For NFAS configurations only, an array of [IISDN_NUM_DS1_INTERFACES] bit masks that identifies the B-channels the host requested in service using the *L4L3mENABLE_PROTOCOL* or *L4L3mENABLE_B_CHANNEL* message. B-channels are numbered 1 - 23; 24 is reserved for the D-channel. For non-NFAS configurations, set to 0x00000000. Possible values for each bit position include:

B-channel is not active. 0x00000000

B-channel is active. 0x00000001

nfas_primary_dchan_status

NFAS Primary D-channel Status. Indicates the current status of the primary D-channel. If NFAS is not configured, set to 0x00. Possible values include:

IISDNdcsIS 0x02

IISDN primary D-channel is in service.

IISDNdcsOOS 0x03

Primary D-channel is out-of-service.

IISDNdcsMOOS 0x04

Maintenance out-of-service state not currently supported.

IISDNdcsMB 0x05

Primary D-channel is in maintenance busy state.

Output

None.

Details

Use *L3L4mPROTOCOL_STATUS* instead of *L3L4mD_CHANNEL_STATUS* (a deprecated function in *Volume 6, Appendix H*).

The *L3L4mPROTOCOL_STATUS* message is in response to any of the following conditions:

- Receipt of an *L4L3mENABLE_PROTOCOL* message.
- Receipt of an *L4L3mREQ_PROTOCOL_STATUS*.
- Change in link status; the *L4_ref* in this instance is 0xFFFF.
- Layer 2 error reported in *l2_detail*; *l2_detail* bit set in the IISDN_LEVEL2_CNFG structure for this D-channel.
- Receipt of an ISDN RESTART message specifying to drop all calls associated with this D-channel.

The remainder of these details pertain to ISDN applications only.

In NFAS configurations, the protocol reports status for the primary D-channel. Additional status indicators represent D-channel states unique to NFAS configurations. The primary D-channel has a status of either in service or out of service.

The message also reports D-channel restart conditions in NFAS. In this case, calls associated with a D-channel might be lost according to the following general rule:

- RESTART condition – all calls on associated B-channels are lost

In addition to D-channel status, the message contains the current status (in service or not active) and the host requested status for all B-channels associated with the D-channel. In NFAS configurations, the message generates this D-channel status as two arrays, each containing a number of bit masks equal to the number of interfaces.

L3L4mRAW_QDATA

Purpose

Passes undecoded Q.931 packets from the network to the host.

Message

IISDN_RAW_QDATA *raw_q931_data*

Message ID

0x16

Input Fields

```
unsigned short len;
unsigned char qdata[IISDN_NMAX_RAWQ];
```

Input

len

Length. Number of bytes in the data string that follows this short integer. Possible values are up to that defined in IISDN_MAX_RAWQ.

qdata

Raw Q.931 Data. String of up to IISDN_MAX_RAWQ bytes containing raw Q.931 data as received over the D-channel. The number of bytes in this string must match the value specified in *len*.

Output

None.

Details

The protocol generates this message only if *L4L3mENABLE_PROTOCOL* specifies the IISDN_LEVEL3_CNFG structure *append_raw_qmsg* field that set up ISDN D-channel processing.

In raw Q.931 message mode, either one or two messages are sent to the host for every Q.931 message received from the network over the D-channel. If the Q.931 message is one that the Dialogic® Brooktrout® firmware can process, the first is the standard decoded message (such as *L3L4mSETUP_IND*). This message is immediately followed by the *L3L4mRAW_QDATA* message containing the undecoded Q.931 packet. Note that these messages generate only one interrupt.

If the message is standard by not one normally processed by the Dialogic® Brooktrout® firmware, the undecoded information is passed to the host in a single *L3L4mRAW_QDATA*. This allows applications to make use of proprietary or site-specific messages. The host is then responsible for any processing required.

Note: Messages received in this way have no effect on ISDN call state.

L3L4mRESTART

Purpose

Signifies receipt of a Q.931 RESTART message from the network.

Message

None.

Message ID

`0x0D`

Output

The host checks any data it keeps on active calls. If there were active calls on any channel that restarted, the host updates its data to indicate the calls no longer exist.

L3L4mSETUP_IND

Purpose

Indicates an incoming call request received by the Dialogic® Brooktrout® module.

Message

IISDN_SETUP_DATA *setup_data*

Message ID

0x01

Input Fields

```
unsigned long call_type;
IISDN_CALLING_INFO calling_party;
IISDN_CALLED_PARTY called_party;
IISDN_REDIREC_NUM redirect_num;
unsigned char net_spfc;
unsigned short bc_len;
unsigned char bearer_cap[IISDN_MAX_BC];
IISDN_USER_INFO user_info;
IISDN_Q922_DLCI fr_dlci;
unsigned char feature_availability;
unsigned char bearer_selection;
unsigned char sending_complete;
IISDN_PROGRESS_IND progress_ind;
IISDN_ORIG_CALLED_NUM orig_called_num;
```

Input

call_type

Call Type. Bit mask identifying what type of call should be established. Most common value is 0x00000001 (normal voice call). Possible values include:

IISDNcalltypVOICE 0x00000001

Normal voice call.

IISDNcalltypMODEM 0x00000002

3.1kHz audio.

IISDNcalltyp56K 0x00000004

56K data call, unknown type.

IISDNcalltyp64K 0x00000008

64K data call, unknown type.

IISDNcalltyp64K_REST 0x00000010

Restricted 64K data call.

IISDNcalltyp384K 0x00000020

384K data call, unknown type.

calling_party

Calling Party Structure. See *Calling Party (IISDN_CALLING_PARTY) on page 857*.

called_party

Called Party Structure. See *Called Party (IISDN_CALLED_PARTY) on page 855*.

redirect_num

Redirecting Number Structure. See *Redirecting Number (IISDN_REDIRECT_NUM) on page 871*.

net_spfc

Call-by-Call Feature. Indicates the network-specific call type. Generally, the non-specific default value is used. If another value is specified, the PRI line purchased from the service provider must be configured to support that call type.

IISDNnsNULL 0x00

Default value; type not specified.

IISDNnsATT_SDN *or* IISDNnsNTI_PRIVATE 0x01

AT&T Software Defined Network or Nortel Private Network.

IISDNnsATT_MEGACOM 800 *or* IISDNnsNTI_INWATS 0x02

AT&T Megacom 800 Service or Nortel InWATS.

IISDNnsATT_MEGACOM *or* IISDNnsNTI_OUTWATS 0x03

AT&T Megacom 800 Service or Nortel OutWATS.

IISDNnsNTI_FX 0x04

Nortel Foreign Exchange.

IISDNnsNTI_TIE_TRUNK 0x05

Nortel Tie Trunk.

IISDNnsATT_ACCUNET 0x06

AT&T Accunet.

IISDNnsATT_I800 0x08

AT&T International 800 Service.

IISDNnsATT_MULTIQUEST *or* IISDNnsNTI_TRO 0x10

Nortel TRO Call.

bc_len

Length of Bearer Capability Information Element (IE). When value is not zero, indicates an undecoded Bearer Capability IE is included in this message. For more information on Bearer Capability and IEs, refer to the specification appropriate to your ISDN service.

bearer_cap[IISDN_MAX_BC]

Bearer Capability Information Element (IE). If the previous field is a value other than zero, this array contains an undecoded Q.931 Bearer Capability IE. For more information on Bearer Capability and IEs, refer to the specification appropriate to your ISDN service.

Packet size is 128 bytes. 0x80

Packet size is 256 bytes. 0xFF

user_info

User Info Structure. See *User Info (IISDN_USER_INFO) on page 875*.

fr_dlci

Q.933 DLCI Negotiation Structure. See *Q.933 DLCI Negotiation (IISDN_Q922_DLCI) on page 870*.

feature_availability

Feature Availability. Indicates what special features are available for this call.

IISDNfaNONE 0x00

No special features available for this call.

IISDNfaFLEXIBLE_BILLING 0x01

AT&T Variabill (flexible billing) feature available for this call.

bearer_selection

Bearer Channel Selection. Used for B-channel negotiation feature. Indicates if B-channel negotiation is enabled by setting the IISDNctB_CHAN_NEGOT bit in the *L4L3mENABLE_PROTOCOL* *conn_type* field.

IISDNbsNONE 0x00

B-channel negotiation is not configured.

IISDNbsPREFERRED 0x01

B-channel is preferred; host must respond with B-channel and interface number (if NFAS) in its next message to the Dialogic® Brooktrout® module.

IISDNbsEXCLUSIVE 0x02

B-channel is exclusive; host must respond with B-channel and interface number (if NFAS) in its next message to the Dialogic® Brooktrout® module.

sending_complete

This is a flag to indicate that sending is complete on the Info Element message. If a 0 appeared, this indicates that no sending completed. A non-zero indicates the message sending completed.

progress_ind

Progress Indication Structure. See *Progress Indication (IISDN_PROGRESS) on page 867*.

orig_called_num

Specifies the appropriate number of digits. A value of 0 designates the feature is unavailable.

Output

To reject a call, the host sends an *L4L3mCLEAR_REQUEST*. Otherwise, messages are sent as follows:

ISDN Calls

L4L3mPROGRESS_REQUEST or *L3L4mALERTING* followed by *L3L4mCONNECT*

ISDN with Fast Connect Calls

L4L3mCONNECT_REQUEST

ISDN with B-Channel Negotiation

The host must assign the *L4_ref* in the responding message.

The module automatically generates a CALL PROCEEDING message to the network after receiving of an incoming SETUP message in the following cases:

- B-channel negotiation is *not* enabled.
- To enable B-channel negotiation use *both* the *b_chan_negot* and *proc_on_exclusv* in the IISDN_LEVEL3_CNFG structure of an *L4L3mENABLE_PROTOCOL* message and the requested channel is exclusive.

If just *b_chan_negot* is set, no CALL PROCEEDING message is generated; the host must send the *L4L3mCALL_PROCEEDING_REQUEST* message.

Host response to an *L3L4mSETUP_IND* with an *L4L3mCONNECT_REQUEST* message is restricted to applications where the service is provided by an AT&T 4ESS Fast Connect Feature.

Details

The following information is supplied:

- Type of call (voice or data and protocol being used).
- Calling Party number, if available, up to a maximum of 17 digits.
- Called Party number up to a maximum of 17 digits.
- Redirecting number, if call is forwarded.
- Feature availability (such as AT&T Variabill).
- Feature type for the call.
- Bearer selection information (B-channel negotiation).

Calling Party and Called Party, if used, are specified in *calling_party* and *called_party*.

The host application must save *call_ref* specified in the common header of this message. This value is used for future message processing for this call.

If the system connected to the interface uses the AT&T 4ESS Fast Connect Feature, no PROGRESS or ALERTING messages are required for call connection.

L3L4mSTATUS_IND

Purpose

Indicates that the network generated a STATUS message due to a Q.931 protocol error.

Message

None.

Message ID

`0x07`

Input

The L3L4 common header contains all data for the message.

Output

None. This message is generally logged and ignored, since unrecoverable errors result in the clearing of the call.

L3L4mUNIVERSAL

Purpose

Passes proprietary or other ISDN Q.931/932 messages not supported by Dialogic® Brooktrout® firmware to the host.

Message

IISDN_UNIVERSAL *universal*

Message ID

0x17

Input Fields

```
unsigned char msg_id;
unsigned char ie_count;
unsigned char null_call_ref;
unsigned char q931_escape;
IISDN_IE_STRUCT ie;
```

Input

msg_id

Q.931/932 Message ID. Indicates the type of message being sent. The receiving end should be capable of interpreting this value.

ie_count

IE Count. Indicates the total number of IEs contained in this message.

null_call_ref

Indicates that the received message contained a call reference value of NULL.

q931_escape

Generates a q931 escape message.

ie

IE Structure. Unused in this message.

Output

None.

Details

This feature allows developers to use proprietary or site-specific messages in an application. In using this message, the following rules apply:

- This message has no effect on call state. Additionally, the application must ensure that the actions required by this message are consistent with the current state of the call.
- Dialogic® Brooktrout® firmware does not check the message ID or message content; this is the responsibility of the application designer.

See *L4L3mUNIVERSAL* for information on the HOLD auxiliary state.

L3L4mUSER_INFO

Purpose

Passes up to 130 bytes of information for a call over the network.

Message

IISDN_USER_INFO *user_info*

Message ID

0x09

Input Fields

```
unsigned short len;
unsigned char info{IISDN_MAX_USER_INFO];
IISDN_USER_INFO user_info;
```

Input

len

A value of 0 determines that no information is passed.

info{IISDN_MAX_USER_INFO]

The user information.

user_info

User Information Structure. See *User Info (IISDN_USER_INFO) on page 875*.

Output

None.

Details

The message consists of the structure documented in *User Info (IISDN_USER_INFO) on page 875*.

Normally the *L3L4mUSER_INFO* message pertains to a call identified by the *call_ref* and *L4_ref* values. *L3L4mUSER_INFO* to *call_ref* 0x00 is a proprietary feature. If the network sends an ISDN message with a call reference value of zero, it sends the *call_ref* to the host using this message.

28 - B-Channel and D-Channel Maintenance

This chapter describes a Dialogic® Brooktrout® module's ISDN B-channel (Bearer) and D-channel (Data) maintenance procedures.

It has the following sections:

- *B-Channel Maintenance*
- *D-Channel Maintenance*

B-Channel Maintenance

The Dialogic® Brooktrout® module's ISDN Bearer Channel maintenance procedures are initiated by the host application through BSMI (BOSTON Simple Message Interface) messages. The host application maintains the state of individual B-channels on the Dialogic® Brooktrout® firmware. The Dialogic® Brooktrout® firmware updates the host application of any B-channel state changes and B-channel maintenance actions initiated by the far end. The Dialogic® Brooktrout® firmware also maintains an internal table of the state that the host requested the B-channels be set to.

ISDN Messages for B-Channel Maintenance

SERVICE and SERVICE_ACKNOWLEDGE maintenance messages are sent to and received from the far end for ATT, NTI, and ISDN variants only. The Dialogic® Brooktrout® module's ISDN uses the following maintenance messages:

- SERVICE

  Used to bring B-channels IS (In Service) or OOS (Out Of Service).
- SERVICE ACKNOWLEDGE

  Used with SERVICE to bring B-channels IS or OOS.
- RESTART

  Used to restart B-channels. Calls on the B-channels are cleared. For National ISDN1/National ISDN2/NET5 variants, the RESTART message also brings B-channels into service.
- RESTART ACKNOWLEDGE

  Used with RESTART to restart a B-channel. For National ISDN1/National ISDN2/NET5 variants, the RESTART ACKNOWLEDGE message also brings B-channels into service.

BSMI Messages For B-Channel Maintenance

SERVICE and SERVICE_ACKNOWLEDGE maintenance messages are sent to and received from the far end for ATT, NTI, and ISDN variants only.

Refer to *Chapter , Host to Module (L4L3m) Messages, on page 1005* and *Chapter , Module to Host (L3L4m) Messages, on page 1086* for more detail on these BSMI messages:

- *L4L3mENABLE_B_CHANNEL*

  The host sends *L4L3mENABLE_B_CHANNEL* to the Dialogic® Brooktrout® module to bring B-channels into service. The *bchannel* in the message specifies the B-channel.

  For NFAS (Non Facility Associated Signaling) configuration, the message specifies the DS1 interface where the B-channel resides. Use *iface* and *n_bchannel* to specify multiple B-channels.

  When the firmware receives *L4L3mENABLE_B_CHANNEL*, it sets the B-channel state to IS in its internal table. It sends out SERVICE messages to the far end for the requested B-channels with Change Status IE set to IS and starts timer T3M1.

- *L4L3mDISABLE_B_CHANNEL*

  The host sends *L4L3mDISABLE_B_CHANNEL* to the firmware to put B-channels OOS. B-channel is specified using *bchannel* in the message.

  For NFAS configuration, the message specifying DS1 where the B-channel resides uses *iface* and specifies multiple B-channels using *n_bchannel*.

  When the firmware receives *L4L3mDISABLE_B_CHANNEL*, it sets the B-channel state to OOS in its internal table. It sends out a SERVICE message to the far end for the requested B-channel with Change Status IE set to OOS and starts timer T3M1.

- *L4L3mRESTART*

  The host sends *L4L3mRESTART* to the module to restart a specified B-channel or interface. The message specifies the B-channel using *bchannel*.

  For NFAS configuration, the message specifying DS1 where the B-channel resides uses *iface* and specifies multiple B-channels using *n_bchannel*.

  When the firmware receives an *L4L3mRESTART* that does not specify B-channel in *bchannel* or *n_bchannel* fields, it sends an interface RESTART to the far end for the specified interface *iface*. If *L4L3mRESTART* specifies B-channel in *bchannel* or *n_bchannel*, it sends a B-channel RESTART to the far end. The firmware starts timer T316 after sending the RESTART message.

  The message clears all the calls on the specified B-channel. For each cleared call the host receives an *L3L4mCLEAR_WITH_RESTART_REQUEST*.

  For all B-channels restarted by sending B-channel RESTART to the far end, the host receives a *L3L4mB_CHANNEL_STATUS* with *b_channel_status* set to IISDNbcsRESTARTING.

  When the far end sends an interface RESTART message, the hose receives a single *L3L4mB_CHANNEL_STATUS* with *b_channel_status* set to IISDNbcsRESTARTING. The message sets *iface* and *bchannel* to a particular interface value received in *L4L3mRESTART*.

- *L3L4mB_CHANNEL_STATUS*

  Using *L3L4mB_CHANNEL_STATUS*, the firmware informs the host application of B-channel status or when sending an interface restart.

When informing the host about interface restarts, the message sets the *bchannel* field to 0.

The *L3L4mB_CHANNEL_STATUS* message to the host indicates the B-channel state in the following cases:

- A valid SERVICE, SERVICE ACKNOWLEDGE is received from the far end.
- For National ISDN1/National ISDN2/ NET5 variant type, a valid B-channel RESTART ACKNOWLEDGE is received.
- For ATT_5ESS/NT_DMS250 switch type & National ISDN1/National ISDN2 variant or for NET5 variant type, a valid B-channel RESTART is received.
- The host sends an *L4L3mRESTART* for restarting B-channels.
- Timer T316 expires.
- For CCITT Layer 3 connection, timer T308 expires.

The firmware also sends *L3L4mB_CHANNEL_STATUS* to the host indicating an interface restart condition when:

- The host sends an *L4L3mRESTART* for initiating interface restart.

- *L3L4mRESTART*

  The firmware sends an *L3L4mRESTART* to the host when it receives a B-channel or interface RESTART from the far end.

  When the far end sends an interface RESTART message, *L3L4mRESTART* sets the *bchannel* and *bchannel_mask* to 0.

  The message sets *iface* to the particular interface RESTART value received from the far end. After receiving the B-channel RESTART message, the *L3L4mRESTART* message sets the *bchannel* or *bchannel_mask* values.‘

Maintenance Procedures

The far end sends and receives SERVICE and SERVICE_ACKNOWLEDGE maintenance messages for the following variants:

- IISDNvarATT_CUSTOM
- IISDNvarNTI_CUSTOM
- IISDNvarNATL_ISDN

Sending SERVICE Message to the Far-End

The firmware sends a SERVICE message to the far end for each B-channel when it receives an *L4L3mENABLE_B_CHANNEL* or *L4L3mDISABLE_B_CHANNEL* from the host for the specified B-channel. There is only one outstanding SERVICE message at a time (a SERVICE message is outstanding when timer T3M1 is running and waiting for SERVICE ACKNOWLEDGE from the far end). The host requests to set the Change Status IE in the SERVICE message to the IS or OOS state. The firmware starts timer T3M1 after sending the SERVICE message to the far end for a B-channel.

When message sends a SERVICE ACKNOWLEDGE for a B-channel, the firmware checks its internal table to determine the state (host-requested) of the B-channel. If the internal table shows the state:

- OOS for the B-channel, the firmware sets the B-channel state to OOS.
- A state other than OOS for the B-channel, the B-channel takes the value received from the far end in the Change Status IE in SERVICE ACKNOWLEGE message.

The firmware then sends an *L3L4mB_CHANNEL_STATUS* message to the host indicating the latest B-channel state.

If the service timer T3M1 expires before receiving the SERVICE ACKNOWLEDGE, the SERVICE message for the B-channel is again sent to the far end value and starts the timer T3M1.

SERVICE Message Received from the Far-End

When a SERVICE message is received for a B-channel, the firmware checks its internal table to determine the state (host-requested) of the B-channel. If the internal table shows:

- OOS for the B-channel, the firmware sets the B-channel state to OOS.
- A state other than OOS for the B-channel, the B-channel takes the value received from the far end in the Change Status IE in the SERVICE message.

The firmware sends a SERVICE ACKNOWLEDGE message for the B-channel to the far end with Change Status IE value set to the latest B-channel state.

If the host requested to send a SERVICE message for this B-channel through *L4L3mENABLE_B_CHANNEL* or *L4L3mDISABLE_B_CHANNEL*, the host request is ignored.

Then the firmware sends an *L3L4mB_CHANNEL_STATUS* message to the host indicating the latest B-channel state.

Sending RESTART Message to the Far-End

The firmware sends a RESTART (for an interface or B-channel) message to the far end when it receives an *L4L3mRESTART* message from the host. After receiving the RESTART message from the far end, the restart timer T316 starts.

If the firmware receives a RESTART ACKNOWLEDGE (for interface or B-channel) message from the far end before timer T316 expires, it sends *L3L4mB_CHANNEL_STATUS* to the host for interface/B-channel.

For a National ISDN1/National ISDN2/NET5 switch variant, the host can use RESTART messages to bring up B-channel.

When the firmware receives a B-channel RESTART ACKNOWLEDGE message from the far end a National ISDN1/National ISDN2/NET5 switch variant, it updates the B-channel state to IS. When the firmware receives interface RESTART ACKNOWLEDGE, it sets all the B-channels states on the interface to IS. The firmware also changes the B-channel state to IS in the internal table that maintains the host requested B-channel service states.

When the firmware receives a RESTART ACKNOWLEDGE:

- For the B-channel, it sends the host the latest B-channel state using *L3L4mB_CHANNEL_STATUS*.
- For an interface, the firmware sends to the host an *L3L4mPROTOCOL_STATUS* message with *b_channel_service_state* bitmask for the interface set to state of B-channels on the interface.

If the restart timer T316 expires before the firmware receives any RESTART ACKNOWLEDGE message from the far end for the B-channel or interface and Layer 3 connection type is not CCITT (*ccitt_mode* = 0 in *L4L3mENABLE_PROTOCOL*), a RESTART message is again sent to the far end and starts the timer T316.

RESTART Message Received from the Far-End

When the firmware receives a RESTART message (for B-channel/Interface/Group Restart) from the far end, it sends an *L3L4mRESTART* (for B-channel/Interface restarts only) to the host.

The firmware sends *L3L4mCLEAR_WITH_RESTART_REQUEST* to the host for all calls in the affected B-channel (for B-channel /Interface/Group Restart).

The firmware then sends a RESTART ACKNOWLEDGE (B-channel/Interface/Group Restart Acknowledge) message to the far end.

When firmware receives B-channel RESTART message from the far end for National ISDN1/National ISDN2 variants or for NET5 variant, it takes the following action:

- The firmware updates its internal table that maintains the host requested B-channel state to show the host requested the B-channel be In Service (IS). The host is responsible for taking any B-channel out of service that it wanted OOS.
- The firmware sets the B-channel state to IS and sends an *L3L4mB_CHANNEL_STATUS* message to the host with latest B-channel status. It then sends an *L3L4mRESTART* message to the host and a B-channel RESTART ACKNOWLEDGE message to the far end.

When the firmware receives Interface RESTART message from the far end for National ISDN1/National ISDN2 or NET5 variant, it takes the following action:

- The firmware updates its internal table that maintains the host requested B-channel state to reflect as if host requested all the B-channels on the interface be In Service (IS). The host is responsible for taking any B-channel out of service that it wanted OOS.
- Firmware sends an *L3L4mPROTOCOL_STATUS* message to the host with updated *b_channel_service_state* for the interface. It then sends an *L3L4mRESTART* message to the host and a RESTART ACKNOWLEDGE message to the far end.

When the firmware receives Group RESTART message from the far end for National ISDN1/National ISDN2 variants, it takes the following action:

- The firmware updates its internal table that maintains the host requested B-channel state to show the host requested all the B-channels on all 20 interfaces in the NFAS group be In Service (IS). The host is responsible for taking any B-channel Out Of Service that it wanted OOS.
- Firmware sends an *L3L4mPROTOCOL_STATUS* message to the host with *b_channel_service_state* updated for all 20 interfaces on the NFAS group. It then sends a RESTART ACKNOWLEDGE message to the far end.

D-Channel Maintenance

This section describes the role of the host application when establishing and releasing the Dialogic® Brooktrout® module's ISDN D-channel and when interacting with B-channel maintenance procedures.

BSMI Messages for D-Channel

For more detail on these BSMI messages, please refer to:

Chapter , Host to Module (L4L3m) Messages, on page 1005

Chapter , Module to Host (L3L4m) Messages, on page 1086

- *L4L3mENABLE_PROTOCOL*

  The host application sends *L4L3mENABLE_PROTOCOL* to firmware to establish the D-channel.

  The field *b_channel_service_state* in *L4L3mENABLE_PROTOCOL* is set to the bitmap that specifies which B-channels on a specific interface are brought into service.

  The firmware uses the specific *b_channel_service_state* value to update its host-requested B-channel service state internal table.

  When the host wants to reestablish or release a D-channel LAP-D connection, it sends *L4L3mENABLE_PROTOCOL* with *epcmd* set to IISDNepcmdDL_ESTABLISHED or IISDNepcmdDL_RELEASE. The state of B-channels is not affected when firmware receives the *epcmd* value.

- *L4L3mDISABLE_PROTOCOL*

  The host sends *L4L3mDISABLE_PROTOCOL* to the firmware to close a D-channel.

- *L4L3mREQ_PROTOCOL_STATUS*

  The host application sends *L4L3mREQ_PROTOCOL_STATUS* to the firmware to request information about the state of D-channels or B-channels.

- *L3L4mPROTOCOL_STATUS*

  The firmware sends *L3L4mPROTOCOL_STATUS* to the host application with information about the state of D-channels or B-channels.

  For non-NFAS configurations, *status* and *channels* show the state of D-channel and B-channels respectively. For NFAS configuration, the *n_b_chan_req* field shows which B-channels the host application requested set to IS.

  The *n_b_channels* field shows the B-channels that are actually IS in the firmware.

  The nfas_primary_dchan_status field shows the status of the primary D-channel in the firmware.

  The firmware sends *L3L4mPROTOCOL_STATUS* to the host for the following conditions:

  - In response to *L4L3mENABLE_PROTOCOL* from the host.
  - When the state of the D-channel changes in the firmware.
  - In response to *L4L3mREQ_PROTOCOL_STATUS* from the host.
  - For an ATT_5ESS or NT_DMS250 switch and National ISDN1 or National ISDN2 variant, in response to an Interface/Group RESTART from the far end.
  - For a NET5 variant, in response to an Interface RESTART from the far end.
  - For a Switch Variant of National ISDN1, National ISDN2, or NET5, in response to an interface RESTART ACKNOWLEDGE from the far end.

Volume 6 - Appendices

About this Volume

Volume 6, Appendices, is a grouping of appendices that relate to the reference material in Volumes 1 through 5, including:

- Configuration files
- Bfv API structures
- Hangup codes
- BSMI and ISDN cause codes
- Infopkt parameter values
- Call Progress notes
- Country-specific parameter files
- Deprecated and unsupported functionality
- SR140 security capabilities

A - Configuration Files

This appendix describes configuration files, defining their structure and providing instructions for creating each of them.

It has the following sections:

- ***User-Defined Configuration File***

  The user-defined configuration file (*btcall.cfg*) contains configuration parameters for the Bfv API and driver. See *page 1152*.

- ***Call Control Configuration File***

  The call control configuration file (*callctrl.cfg*) contains configuration parameters that define how the user wants the Bfv API to configure the modules for call control. See *page 1170*.

  The *callctrl.cfg* file replaces the *teleph.cfg* and *ecc.cfg* files previously used to define the call control configuration. See *page 1478* if your application uses the *teleph.cfg* and *ecc.cfg* configuration files. The Dialogic® Brooktrout® Bfv API no longer supports the *teleph.cfg* and *ecc.cfg* files so the developer must convert the application to use the new call control configuration parameters.

- ***Routing Table Configuration File***

  The routing table configuration file contains configuration parameters that define one or more routing rules which specify how the user wants to route inbound calls to specific SR140 channels based on information associated with the calls. See *page 1322*.

Note: Inbound call routing is only supported on the SR140.

- ***SRTP Configuration File***

  The SRTP configuration file section describes the content of the SRTP configuration file. This file is an ASCII file that contains key parameters and values used to configure SRTP. See *page 1331*.

- ***TLS Configuration File***

  The TLS configuration file section describes the content of the TLS configuration file. See *page 1334*.

- ***Parameters for Technical Support Purposes***

  These parameters might be specified for the user-defined configuration file when seeking assistance from Dialogic Technical Services and Support. See *page 1337*.

User-Defined Configuration File

The user-defined configuration file contains parameters that set values such as specific fax formatting. The Bfv API supplies a default configuration file named *btcall.cfg* in the *app.src* directory. The programs in *app.src* use *btcall.cfg*.

You can edit the *btcall.cfg* file with a standard text editor, or Windows users can edit this file using the Dialogic® Brooktrout® graphical configuration tool.

For users who have a *btcall.cfg* file created for a release of the Bfv API prior to Brooktrout SDK 3.1, delete the following parameters from the file. These parameters have been removed or moved to another configuration file as indicated:

Parameter	Description
did_digits	This DID digit detection parameter has been modified and moved to the *callctrl.cfg* file.
did_variable	This DID digit detection parameter has been modified and moved to the *callctrl.cfg* file.
digital	The call control configuration file (*callctrl.cfg*) replaces the configuration file defined by the *digital* parameter.
isdn	The call control configuration file (*callctrl.cfg*) replaces the configuration file defined by the *isdn* parameter.
line_encoding	The call control configuration file (*callctrl.cfg*) that replaces the *teleph.cfg* file.
nrings	This parameter has been renamed *num_rings* and moved to the *callctrl.cfg* file.
switch_hook	This parameter has been renamed *flash_hook_duration* and moved to the *callctrl.cfg* file.
teleph	The call control configuration file (*callctrl.cfg*) replaces the configuration file defined by the *teleph* parameter.

The application passes the *btcall.cfg* file to the *BfvLineReset* function which uses the information to reset and initialize the system. The Bfv API assumes that the *btcall.cfg* file resides in the current directory unless you specify a path with the filename. Different user configuration files can be stored and passed to the *BfvLineReset* function as needed. The *BfvLineConfig* function can also be used for configuration.

Parameters can be listed in any order and typed in either uppercase or lowercase or both. Only one parameter per line is permitted. Parameters must be separated from their values – a decimal integer, a hexadecimal integer, or a character string – by one or more spaces. Commas, colons, and dashes are not valid parameter separators. The default value is automatically supplied for each missing parameter; and parameters that do not match any of the valid keywords are ignored. If a parameter appears more than once, the last occurrence is the one that takes effect.

A parameter applies to all product types unless its description lists a specific product type. For convenience only, the list of parameters on the following pages groups the parameters for answer machine detection at the beginning.

Note: The parameters for answer machine detection only apply to applications for the Microsoft Speech Server. These parameters are not supported by the Bfv API in this release.

Normally, whitespace characters (spaces and tabs) should not be included. To include whitespace, the character string can begin and end with double quotes (").

The Bfv API treats any line that begins with the # character as a comment and ignores that line. All character strings that represent filenames must consist of printable ASCII characters.

Parameter	Purpose
Standard btcall.cfg Parameters	
agc	Specifies the automatic gain control (AGC) method to use during speech recording. Does not apply to full duplex recording or for use with ASR. `0` None. `>0` Dynamic AGC; levels adjusted during recording. *Value Type:* decimal *Default:* `1`
badline_behavior	Specifies the behavior during non-ECM when a bad line is detected in received MH or MR data. Set this parameter to: `0` Replace with last good line. `1` Drop the line. `2` Attempt to repair the line (not available on TruFax®). *Value Type:* decimal *Default:* `0`
bt_cparm	Specifies the path and name of the country telephony parameter file to use. *Value Type:* character string *Default:* *BT_CPARM.CFG*
busy_dt_ct	Specifies the number of consecutive `BUSY1`, `BUSY2`, `ROBUSY`, or `DIALTON` call progress values that must occur before *BfvLineOriginateCall* terminates with that result. *Value Type:* decimal *Default:* `1`
call_control	Specifies the name of the call control configuration file to use (see *page 1170*). **Note:** The *callctrl.cfg* file replaces the *teleph.cfg* and *ecc.cfg* files. *Value Type:* character string *Default:* *callctrl.cfg*
ced_timeout	Specifies the length of time, in 10 ms units, to wait for a fax answer tone (CED tone) from a remote fax machine. This parameter can only be set if the host country permits changing the *wait_for_ced_high* and *wait_for_ced_low* (see *page 1439*) timeout values. The *ced_timeout* parameter also controls the amount of time CNG plays. Maximum value is 65535 (655 seconds). *Value Type:* decimal *Default:* Country dependent; 4000 (40 secs) in the USA

Parameter	Purpose

channel	Specifies a channel number value that allows the user to apply a user-defined configuration file keyword only to this specific channel. Any other channel encountering the specified keyword ignores it. Set as follows:
	`channel chan_num keyword [keyword_params]`
	`chan_num` — Specifies the ordinal channel that applies the keyword.
	`keyword` — Specifies the keyword to apply.
	`[keyword_params]` — Specifies optional parameters to the keyword.
	Example:
	`channel 8 id_string +1-408-370-1171`
	This example configures channel 8 with the number of a fax machine.
	Restricted Keywords:
	The following keywords in this user-defined configuration file (*btcall.cfg*) cannot be selected:
	call_control
	channel
	font_file
	pcpm_table
	tone/pulse (see Note)
	Note: When using the *btcall.cfg* file on a per channel basis, the dialing type specified for the first channel to execute dictates how the other channels dial no matter how they are configured. This situation only occurs when more than one channel executes concurrently. If channel one dials dtmf digits first and channel two follows but is configured for pulse dialing, channel two dials dtmf digits too. Use the P or T character in the dial string of the call to override pulse or tone default dialing modes.
	Value Type: character string

country_code	Specifies the international country code with modifiers. Initial digits (up to 3) identify the host country; the last digit supplies a modifier for properties such as the phone system attached to the board. The *ccode.h* header file contains the available country codes. *Value Type:* hexadecimal *Default:* 0010 (USA)

Parameter	Purpose
debug	Turns on debug mode. The presence of this parameter turns on Bfv API debug mode using the *BfvDebugModeSet(DEBUG_ALL)* function call after the first call to *BfvLineReset*. The Bfv API only uses this setting if the application did not already enable debug mode. You can also specify up to two optional filenames and a maximum file size, if the application did not previously set a debug function. See *Volume 1*, *BfvDebugFuncSet*, *BfvDebugModeSet* and *BfvDebugModeSetAdv* functions for more detailed information. The format for setting the parameter is: `debug [fname [fname2 max_size]]` Setting this parameter provides the following options: ■ No filename — Turns on debug mode without naming a file, sending the output to the standard output device. ■ One filename — Turns on debug mode, sending the output to the filename specified. ■ Two filenames — Turns on debug mode, sending the output to the first filename (`fname`) until the output exceeds the size in bytes specified by `max_size`. At this point, the output goes to the second filename (fname2) until it exceeds the maximum size. Then the process repeats, going back to writing the output to the first filename. Different processes using the same filename at the same time might result in overwriting the file and creating an incomplete debug output. If you include "%p" in the filename, the Bfv API inserts the process ID at that point in the name to provide unique file creation. *Value Type:* character string *Default:* disabled, stdout
debug_options	Specifies a bit-mapped value that allows the user to define a set of debug options. The format for setting the parameter is: `debug_options <option_value>` Defines a bit-mapped value specifying the selected debug options. Set bit values as follows: Bit 0 — Causes the Bfv API to trace each function entry and exit point in the Bfv API. Bit 1 — When set in conjunction with bit 0, causes the Bfv API to trace each function entry and exit point in the Bfv API and list the arguments and return values for each function. *Default:* `0` (no debug options)

Parameter	Purpose
dtmf_hi_to_lo_twist_idle	Specifies the maximum amplitude allowed for a digit's high frequency to be louder than its low frequency when detecting DTMF digits while not playing speech. Set this parameter to: 0 No limit 1 4.2 dB 2 5.2 dB 3 6.7 dB 4 8.2 dB 5 10.2 dB 6 13.2 dB 7 18.2 dB *Range:* 0 – 7 *Value Type:* decimal *Default:* 2 (5.2 dB)
dtmf_hi_to_lo_twist_play	Specifies the maximum amplitude allowed for a digit's high frequency to be louder than its low frequency when detecting DTMF digits during speech playback. Set this parameter to: 0 No limit 1 4.2 dB 2 5.2 dB 3 6.7 dB 4 8.2 dB 5 10.2 dB 6 13.2 dB 7 18.2 dB *Range:* 0 – 7 *Value Type:* decimal *Default:* 4 (8.2 dB)
dtmf_in_to_in_ratio_idle	Specifies the minimum difference required between the digit's highest signal energy and its next highest signal energy when detecting DTMF digits while not playing speech. *Unit:* 0.5 dB *Range:* 2 – 20 (1 – 10 dB) *Value Type:* decimal *Default:* 16 (8 dB)

Parameter	Purpose
dtmf_in_to_in_ratio_play	Specifies the minimum difference required between the digit's highest signal energy and its next highest signal energy when detecting DTMF digits during speech playback. *Unit:* 0.5 dB *Range:* 2 – 20 (1 – 10 dB) *Value Type:* decimal *Default:* 6 (3 dB)
dtmf_in_to_out_ratio_idle	Specifies the minimum difference required between the digit's signal energy and noise energy when detecting DTMF digits while not playing speech. *Unit:* 0.5 dB *Range:* 1 – 10 (0.5 – 5 dB) *Value Type:* decimal *Default:* 8 (4 dB)
dtmf_in_to_out_ratio_play	Specifies the minimum difference required between the digit's signal energy and noise energy when detecting DTMF digits during speech playback. *Unit:* 0.5 dB *Range:* 1 – 10 (0.5 – 5 dB) *Value Type:* decimal *Default:* 2 (1 dB)
dtmf_lo_to_hi_twist_idle	Specifies the maximum amplitude allowed for a digit's low frequency to be louder than its high frequency when detecting DTMF digits while not playing speech. Valid values are: 0 No limit 1 6.7 dB 2 8.2 dB 3 9.2 dB 4 10.7 dB 5 12.7 dB 6 15.2 dB 7 18.2 dB *Range:* 0 – 7 *Value Type:* decimal *Default:* 2 (8.2 dB)

Parameter	Purpose
dtmf_lo_to_hi_twist_play	Specifies the maximum amplitude allowed for a digit's low frequency to be louder than its high frequency when detecting DTMF digits during speech playback. Valid values are: 0 No limit 1 6.7 dB 2 8.2 dB 3 9.2 dB 4 10.7 dB 5 12.7 dB 6 15.2 dB 7 18.2 dB *Range:* 0 – 7 *Value Type:* decimal *Default:* 3 (9.2 dB)
dtmf_min_off_idle	Specifies the minimum silence required to determine that a digit has ended when detecting like DTMF digits while not playing speech. Specify values for this parameter in 15-ms increments as shown in the supported values list. If you specify a value that is not included in the supported values list, the Bfv API rounds the value down to the lowest supported value. For example, specifying a value of 44 results in an actual value of 30. *Unit:* ms *Legal Values:* 30, 45 *Value Type:* decimal *Default:* 45
dtmf_min_off_play	Specifies the minimum silence required to determine that a digit has ended when detecting like DTMF digits during speech playback. Specify values for this parameter in 15-ms increments as shown in the supported values list. If you specify a value that is not included in the supported values list, the Bfv API rounds the value down to the lowest supported value. For example, specifying a value of 44 results in an actual value of 30. *Unit:* ms *Legal Values:* 30, 45 *Value Type:* decimal *Default:* 45

Parameter	Purpose
dtmf_min_on_idle	Specifies the minimum duration that a digit must be present in order to be detected while not playing speech. Specify values for this parameter in 15-ms increments as shown in the supported values list. If you specify a value that is not included in the supported values list, the Bfv API rounds the value down to the lowest supported value. For example, specifying a value of 44 results in an actual value of 30. *Unit:* ms *Legal Values:* 30, 45, 60, 75, 90 *Value Type:* decimal *Default:* 30
dtmf_min_on_play	Specifies the minimum duration that a digit must be present in order to be detected during speech playback. Specify values for this parameter in 15-ms increments as shown in the supported values list. If you specify a value that is not included in the supported values list, the Bfv API rounds the value down to the lowest supported value. For example, specifying a value of 44 results in an actual value of 30. *Unit:* ms *Legal Values:* 30, 45, 60, 75, 90 *Value Type:* decimal *Default:* 45
ecm_enable	Turns ECM (error correction mode) on or off. If disabled, MMR fax compression on the line is unavailable. The normal ECM frame size is 256 bytes. You can enable a frame size of 64 bytes, but the channel uses that frame size on transmit only. On receive, it always uses the frame size the transmitter selects. TruFax® BRI boards do not support ECM. Disabling ECM applies to all other board types. T.38 Internet Aware Fax (IAF) modulation requires ECM to be enabled. 0 Turns ECM off. 1 Turns on ECM, 256-byte frames. 2 Turns on ECM, 64-byte frames. *Value Type:* decimal *Default:* 1

Parameter	Purpose
eff_pt_caps	Specifies the enhanced fax format page types that the channel is permitted to receive. Values are formed by logically ORing together the base values: `0` Enhanced fax format reception disabled. `1` JPEG. `2` Full color mode (JPEG). `4` Reserved for Huffman tables, do not use. `8` 12 bits/pel, otherwise 8 bits/pel (JPEG). `10` No subsampling (JPEG). `20` Custom illuminant (JPEG). `40` Custom Gamut (JPEG). `100` JBIG. `200` L0 Mode (JBIG). If EFF page reception is enabled, then ECM is automatically enabled for receive faxes regardless of the setting of *ecm_enable*. Not available on TruFax®. *Value Type:* hexadecimal *Default:* `0`
error_mult	Specifies an error multiplication value used to determine if the error percentage on a received page is too high. The number of errors per page is multiplied by this number and the product is divided by 2. If this result exceeds the number of lines on the page, the error percentage per page is too high and an RTN signal is returned to the transmitting station. The value set for this parameter should normally be less than that of the *error_mult_rtp* parameter (corresponding to a larger percentage). The RTN threshold takes precedence over the RTP threshold. *Value Type:* decimal *Default:* `40` (for a 5% error rate)

Parameter	Purpose
error_mult_rtp	Specifies an error multiplication value used to determine if the error percentage on a received page is too high. The number of errors per page is multiplied by this number and the product is divided by 2. If this result exceeds the number of lines on the page, the error percentage per page is too high and an RTP signal is returned to the transmitting station. The value set for this parameter should normally be greater than that of the *error_mult* parameter (corresponding to a smaller percentage). The RTN threshold takes precedence over the RTP threshold. *Value Type:* decimal *Default:* `200` (for a 1% error rate)
error_thresh	Specifies an error threshold value of n (2n for fine resolution) number of consecutive bad G3 lines on a received page. A page with errors in this number of consecutive lines is considered bad, regardless of the results from *error_mult*. An RTN is returned when a “bad” page occurs. *Value Type:* decimal *Default:* `3`
fax_rtp_enable	Controls whether an RTP FSK signal can be sent during fax receive. Set this parameter to: `0` Disabled `1` Enabled *Value Type:* decimal *Default:* `1` (enabled)
flag_timer	Specifies the maximum time to wait for a HDLC frame after detecting flags for V.21 data. *Unit:* ms *Range:* maximum value is 65535 *Value Type:* decimal *Default:* `3000`

Parameter	Purpose
font_file	Specifies the name of the file that contains the transmit/convert font for ASCII. An optional font number, indicating the downloadable font to use, can be specified (if no font number is specified, `0` is assumed). The font file must be located in the current directory, or the correct path must be included with its name. The file is opened, and the contents downloaded to the module when *BfvLineReset* is called using the *mill_load_fonts* option. Multiple occurrences of font file parameters with different font numbers are permitted in the configuration file. When a font number that is specified for ASCII conversion has not been downloaded, a default font is used. This is font `255`. Font 255 may be specified using the *font_file* keyword. If not, it defaults to `ibmpcps.fz8` (no path). When font downloads are done as described above, font 255 is always downloaded regardless of whether other font numbers are listed using this keyword. Some font numbers may be reserved for preloaded fonts. *Range for font number:* `0 – 6,255` *Value Type:* character string; decimal can be included and is optional *Default:* `ibmpcps.fz8` (no path) and `0`
iaf_enable	Enable (1) or disable (0) T.38 Internet Aware Fax (IAF) modulation. Error Correction Mode (ECM) must be enabled for IAF. In addition to this, the T.38 version (*t38_fax_version*) specified in the Call Control configuration file must be set to 1 or higher for IAF. The IAF Maximum Bit Rate parameter (*iaf_max_bit_rate*) should be used to specify the maximum bit rate when IAF is enabled. This parameter applies to the SR140 only. The SR140 must have an IAF license. *Value Type:* decimal *Default:* 0 (disabled)
iaf_max_bit_rate	Specifies the T.38 Internet Aware Fax (IAF) Maximum Bit Rate. A value of 0 will choose the default value. The IAF capability parameter (*iaf_enable*) should be set to enable. Error Correction Mode (ECM) must be enabled for IAF. Additionally, the T.38 version (*t38_fax_version*) specified in the Call Control configuration file must be set to 1 or higher for IAF. This parameter applies to the SR140 only. The SR140 must have an IAF license. *Unit:* bits per second *Range:* 14400 to 2400000 *Value Type:* decimal *Default:* 0 (Unlimited up to the IAF license value)

Parameter	Purpose
id_string	Sets the default ID string (up to 20 characters long) for fax machines. The parameter can be overridden by the *BfvFaxSetLocalId* function if the host country permits changing the ID string. *Value Type:* character string *Default:* 20 spaces
line_compression	Specifies the permitted compression types for fax transmission or reception on the phone line. This specification is independent of the file format specified for transmission or reception. If ECM is disabled, then MMR fax compression on the line is unavailable. Valid values are: `0` MH only `1` MR or MH `5` MMR, MR, or MH The TruFax® board only supports option 0 (MH only) and option 1 (MR or MH). Default = `1`. *Value Type:* decimal *Default:* `5`
max_pagelist	Specifies the maximum number of pages allowed for storing results during a call. The last *max_pagelist* `PAGE_RES` structures are accessible via the `FAX_RES` structure if this feature has been enabled. See the `LINE_FAX_RES` macro. *Value Type:* decimal *Default:* `30`
max_timeout	Specifies the maximum time for the Bfv API to delay waiting for activity to occur on a channel. If any Bfv API function allows specification of a timeout value, that value takes precedence. *Unit:* seconds for values <1000 milliseconds for values 1000 and greater *Range:* `0` to `2000000000`, `0` = disabled *Value Type:* decimal *Default:* `0` (disabled)
max_width	Sets the maximum page width permitted for fax reception. Valid values are: `0` 215 mm A4 1728 Normal resolution pixels. `1` 255 mm B4 2048 Normal resolution pixels. `2` 303 mm A3 2432 Normal resolution pixels. *Value Type:* decimal *Default:* `0`

Parameter	Purpose
min_length	Specifies the minimum number of lines required for a fax page received in non-ECM mode. The firmware considers any fax page with fewer lines invalid and responds with an RTN signal. *Unit:* 10 scan lines *Range:* 0 – 255 *Value Type:* decimal *Default:* 0
pcpm_table	Specifies the path and name of the PCPM (Programmable Call Progress Monitoring) file that contains the tone table for the channel to use. If the user does not set this parameter, the Bfv API does not use the programmable call progress monitoring feature. Only available on modules with analog or BRI interfaces. *Value Type:* character string *Default:* None
record_beep_dur	Specifies the length of time to play the beep tone before starting to record speech. *Unit:* ms *Range:* 10 – 10000 *Value Type:* decimal *Default:* 500
record_beep_freq	Specifies the frequency of the beep tone that plays before starting to record speech. *Unit:* Hz *Range:* 300 – 2500 *Value Type:* decimal *Default:* 500
reorder_busy_min_dur	Allows you to adjust the lower detection threshold for reorder busy tone. If the duration of the high and low of the tone cadence is less than this value, the cadence will not be reported as reorder busy tone. Use a smaller value than the default to detect a shorter cadence as reorder busy tone. *Unit:* ms *Value Type:* decimal *Default:* 220 **Note:** Large adjustments to the default value are not recommended.

Parameter	Purpose
restrict_res	Specifies allowable resolutions for fax reception. Values are formed by ORing together the following values: `0` 200H x 100V (normal) and 100H x 100V (for JPEG only) `1` 200H x 200V (fine) `2` 200H x 400V `4` 300H x 300V `8` 400H x 400V `10` 300H x 600V `20` 400H x 800V `40` 600H x 600V `80` 600H x 1200V `100` 1200H x 1200V Regardless of the value chosen, 200H x 100V (normal) and 100H x 100V (for JPEG only) is always allowed. The TruFax® board only supports option 0 (200H x 100V only) and option 1 (200H x 200V). No JPEG for option 0. *Value Type:* hexadecimal *Default:* `1`
silcompr_start	Compresses silence at the start of recording to the time specified in milliseconds when silence compression is enabled. *Unit:* ms *Value Type:* decimal *Default:* `500`
silcompr_middle	Compresses silence in the middle of the recording to the time specified in milliseconds when silence compression is enabled. *Unit:* ms *Value Type:* decimal *Default:* `1000`

Parameter	Purpose
subpwdsep	Enables reception of the SUB, PWD, and SEP FSK signals. Applications typically use these signals to direct or validate incoming calls. To form values, OR together the following base values: `0` SUB, PWD, and SEP reception disabled. `1` SEP reception enabled. `2` PWD reception enabled. `4` SUB reception enabled. Not available on TruFax®. *Value Type:* decimal *Default:* `0`
tone/pulse	Specifies the default dialing mode when using either the T1 Robbed Bit Signaling (RBS) protocol or an analog interface. Valid values are: `pulse` Channel uses pulse dialing mode as the default mode. `tone` Channel uses DTMF tone dialing as the default mode. *Note:* Although the default value for the *tone/pulse* keyword applies to all calls, you can override this value on a per call basis by including either the P or T character in the dial string of the call. *Value Type:* character string *Default:* `tone`
t38_v21preamble_repeat	Controls whether T.38 will send additional copies of the V21preamble flag. Sending additional copies increases robustness and is the default behavior but can cause difficulties for some devices. `0` Do not repeat V21preamble. `1` Repeat V21preamble *Value Type:* decimal *Default:* `1`
v_play_gain	Specifies the initial gain value for speech playback. A positive value indicates a step up; a negative value indicates a step down. The gain steps are the same as those controlled by *BfvSpeechModify*. *Unit:* A factor of 2 for each step up or down *Range:* –3 to +3 *Value Type:* decimal *Default:* `0`

Parameter	Purpose
v_timeout	Specifies the maximum time (in seconds) to wait after the last dialed digit for a final call progress result. Use only when you select `CALL_PROTOCOL_VOICE` mode. This parameter only applies to the use of *BfvLineOriginateCall* and *BfvLineOrigCallDB*. *Unit:* second *Value Type:* decimal *Default:* `60`
v34_2400_baud_ctrl	Turns the use of high speed control signaling on (1) or off (0) for V.34. *Value Type:* decimal *Default:* `1` (enabled)
v34_ci_enable	Turns the ability to detect and send the Call Indicator (CI) signal on (1) or off (0) in order to enter V.8 mode and V.34 mode after missing the initial CED tone. *Value Type:* decimal *Default:* `1` (enabled)
v34_enable	Turns V.34 fax modulation capability on (1) or off (0) if the hardware and firmware support it. When V.34 actually occurs and V.34 fax is enabled, ECM is used regardless of the setting of *ecm_enable* (see *page 1160*). Set only for the TR1034 and SR140. *Value Type:* decimal *Default:* `1` (enabled)

Parameter	Purpose
v34_max_bit_rate	Maximum bit rate the V.34 modem can negotiate. Except for the value 0xFF, this parameter overrides the btcparm.cfg max_bitrate parameter.
	`0` 2400 bps
	`2` 4800 bps
	1 7200 bps
	3 9600 bps
	4 12000 bps
	5 14400 bps
	6 16800 bps
	7 19200 bps
	8 21600 bps
	9 24000 bps
	10 26400 bps
	11 28800 bps
	12 31200 bps
	13 33600 bps
	255 Generic; maximum supported by board/firmware
	Value Type: decimal
	Default: `0xFF`
width_res_behavior	Specifies the action taken as a result of page width or resolution mismatches on fax transmission. Does not affect fax reception. Scaling the fax is not available for all combinations of resolution mismatches.
	`0` Terminates the call
	`1` Scales the fax horizontally and vertically
	Value Type: decimal
	Default: `1`

The following sample file does not include all configuration options. See the supplied *btcall.cfg* file, located in the *config* directory.

```
country_code 0010
id_string First_Fax
v_timeout 10
font_file ibmpcps.fz8
max_pagelist 25
```

Call Control Configuration File

The call control configuration file is an ASCII file that contains general PCM configuration parameters for all telephony hardware units and static telephony connections to be formed for all modules. Users of Windows operating systems can use the graphical configuration tool to edit the *callctrl.cfg* file that Dialogic supplies with the Brooktrout SDK (see the *Dialogic® Brooktrout® Fax Products SDK Installation and Configuration Guide* for instructions on how to use this tool).

The *call_control* parameter in the user-defined configuration file (see *page 1154*) specifies the path and filename of the call control configuration file (the Bfv API uses *callctrl.cfg* as the default value). The application uses the *BfvLineReset* function to pass the *btcall.cfg* file that contains all the telephony configuration information (*callctrl.cfg*) for the Dialogic® Brooktrout® modules to the Bfv API. This function uses the information to reset and initialize the system. The *callctrl.cfg* file contains configurations for ISDN layer 1 and layer 2 regardless of the selected protocol.

A sample *callctrl.cfg* file is located in the *config* directory. Separate examples of analog, BRI, E1 ISDN (PRI), and T1 ISDN configuration files start on *page 1296*, and additional sample telephony configuration files are also located in the *config* directory.

This section describes the content of the call control configuration file as follows:

- *Call Control Configuration File Format*
- *Global Options*
- *Global Module Parameters*
- *Clock Configuration Parameters*
- *Port Configuration Parameters*
- *Specific Parameters for Port Configuration*
- *Internet Protocol (IP) Call Control Configuration Parameters*
- *Examples of PSTN Call Control (callctrl.cfg) Files*

Call Control Configuration File Format

The general format of the file is:

Global options

Parameters that select trace options.

```
l314_trace=XXXX
l413_trace=XXXX
api_trace=XXXX
internal_trace=XXXX
host_module_trace=XXXX
ip_stack_trace=XXXX
trace_file=XXXX
max_trace_file_size=XXXX
max_trace_files=XXXX
```

Global module parameters

Parameters that set configuration values for the whole module.

```
[module.#]
   set_api=XXXX
   auto_connect=XXXX
   connections=XXXX
   pcm_law=XXXX
   jate_redial_restriction=XXXX
   jate_emergency_numberN=XXXX
   channels=XXXX
   vb_firm=XXXX (only set if application uses a virtual module)
   enable_static_ring_detection=XXXX
```

Generic clock configuration parameters

Parameters that set the clock configuration for the whole module.

```
[module.#/clock_config]
   clock_mode=XXXX
   clock_source=XXXX
   clock_compatibility=XXXX
   Advanced H.100 and H.110 options can follow here
```

Port configuration parameters

Parameters that identify the port to configure for the selected signaling protocol. Port-specific configuration parameters allow the user to set protocol or interface options depending on the value set in the `port_config` parameter.

```
[module.#/port.#]
   port_config=XXXX
   Port-specific configuration options can follow here
```

IP call control stack parameters for Bfv API

This section includes parameters that identify the internet protocol (IP) call control stack for the Bfv API to use. Dialogic provides an IP call control stack with the Brooktrout SDK or users can provide a third party stack of their own choosing.

The section also includes parameters to define the T.38 fax transport for the Bfv API and a means for the IP call control stack to read custom key-value pairs from the call control configuration file.

```
[host_module.#]
   enabled=XXXX
   module_library=XXXX
[host_module.#/t38parameters]
   The specific T.38 fax transport parameters can go here
[host_module.#/rtp]
   Specific RTP parameters can go here
[host_module.#/parameters]
   Custom key-value pairs can go here
```

Ethernet interface parameters

Parameters that define the module's Ethernet interface.

```
[module.#/ethernet.#]
   dhcp=XXXX
   ethernet_speed=XXXX
   ip_address=XXXX
   ip_addressV6=XXXX
   ip_arp_timeout=XXXX
   ip_broadcast=XXXX
   ip_gateway=XXXX
   ip_interface=XXXX
   ip_interfaceV6=XXXX
   ip_netmask=XXXX
   media_port_min=XXXX
   media_port_max=XXXX
```

IP call control stack parameters for module

Parameters that identify the internet protocol (IP) call control stack for the module to use.

```
[module.#/host_cc.#]
   host_module=XXXX
   number_of_channels=XXXX
```

The format of this call control configuration file allows you to set the port configuration options for multiple ports in a single module and configure multiple modules in a single file. The example formatting of the file uses nesting only to improve the readability of the code — the file does not require use of this technique.

Note: Do not add multiple host module sections using the same Dynamic Link Library (DLL) or Shared Object (SO) file. The DLL and SO will attempt to use the same resources and will conflict with one another.

Specify one host module and set the sip_max_session or h323_max_session to the maximum number of simultaneous calls needed by the application. The host module section defines a pool of SIP or H.323 sessions that are distributed among both TRxStream and SR140 devices.

Global options

The global options set parameters that affect operation of the entire call control component of the Bfv API. These parameters select the type of tracing and indicate where to store the traced data.

module.#

The `module` number refers to the BOSTON module number. Parameter values set for `module.#` configure the whole board or virtual module. Refer to the *Dialogic® Brooktrout® Fax Products SDK Installation and Configuration Guide, Chapter 1* for more details.

module.#/ clock_config

The `module.#/clock_config` section of the configuration file contains both generic clocking parameters and specific clocking parameters depending on the interface connection.

Note: For users porting from a *teleph.cfg* configuration file, the `clock_config` section replaces the configuration information previously provided for Port 0. The *callctrl.cfg* file does not configure Port 0 as the CT bus.

module.#/port.#

The `port.#` of `module.#/port.#` refers to the number of the hardware unit on the module, and the `port_config` parameter beneath (see *page 1171*) refers to the signaling protocol the port uses.

host_module.#

The `host_module.#` section provides parameters to define an internet protocol (IP) call control stack for the Bfv API to use.

This configuration section also allows you to configure:

- T.38 fax transport parameters for the Bfv API.
- RTP parameters for the Bfv API.
- Custom key-value pairs for the IP call control stack to read from the configuration file.

module.#/ethernet.#

The `module.#/ethernet.#` section provides parameters to define an Ethernet interface for a module, allowing support for modules with multiple interfaces.

module.#/host_cc.#

The `module.#/host_cc.#` section provides parameters to define an IP call control stack for the module to use.

All module and port numbers are in hexadecimal. Text strings are not case sensitive and only use ASCII format.

Comment lines in the file should start with a ‘;’ (semicolon) or a ‘#’ (pound or number) symbol.

If you configure module 0, the settings apply to all modules in the system. If you configure module 0 and also configure settings for a specific module, the settings for the specific module apply to its configuration.

Each parameter has a default value that the system uses if you do not specify a value in the configuration file. The Bfv API defined the default values so that the result provides a working configuration if you do not specify values.

Four sections of parameters define the configuration of a module. If your application uses an IP call control stack, the remaining three sections of parameters apply.

The parameter sections include:

- Global options (see *page 1176*)
- Global module parameters (see *page 1178*)
- Clock configuration parameters (see *page 1186*) including parameters that configure the specific clocking parameters depending on the interface connection (see *page 1188*)
- Port configuration parameters (see *page 1190*)
- IP call control stack parameters for Bfv API (see *page 1246*)
- Module's Ethernet interface parameters (see *page 1286*)
- IP call control stack parameters for a module (see *page 1291*)

The following paragraphs define the parameters for these sections.

Global Options

The following parameters affect operation of the entire call control component of the Bfv API. Specify these parameters at the beginning of your call control configuration file (see *page 1298* for an example of a file containing settings for these parameters).

Parameter

l3l4_trace
l4l3_trace
api_trace
internal_trace
host_module_trace
ip_stack_trace

Value

Selects the type of tracing as follows:

`l3l4_trace`	Traces BSMI messages between layers 3 and 4.
`l4l3_trace`	Traces BSMI messages between layers 4 and 3.
`api_trace`	Traces call control activity to and from the Bfv API functions.
`internal_trace`	Traces call control activity in areas not otherwise covered. Dialogic's engineering personnel use this tracing — application developers are not advised to select this type of tracing.
`host_module_trace`	Traces call control activity to and from all host modules defined in your call control configuration file (see *host_module on page 1291*).
`ip_stack_trace`	Traces call control activity to and from all IP stack module libraries defined in your call control configuration file (see *module_library on page 1246*).

Note: Selecting more than one type of tracing at a time results in duplication.

Set any of these six parameters to one of the following values:

`none`	Does not perform a trace operation (default value).
`error`	Detects errors and stores them in the specified `trace_file`.
`warning`	Detects warnings and stores them in the specified `trace_file`.
`basic`	Stores a simplified trace in the specified `trace_file`.
`verbose`	Stores a complete trace of operations in the specified `trace_file`.
Default:	`none` (no trace performed)

Parameter	Value
trace_file	Turns on tracing and reports results to the filename specified for this parameter. To turn off tracing, comment this parameter out.
max_trace_file_size	Specifies the maximum size, in megabytes, allowed for the trace file. If the trace of operations reaches this size, tracing loops back to the start of the file and the continued trace starts overwriting the older trace. `0` Sets the trace file to an unlimited size. *Unit:* Megabytes *Default:* `10`
max_trace_files	Specifies the maximum number of trace files for the Bfv API to retain on the system's file system. When set to a value greater than 1, the Bfv API appends a sequence number extension to the file name, starting at 1. If the number of created trace files exceeds the value set for this parameter, the Bfv API starts deleting files from the lowest numbered trace log until it frees sufficient disk space to store the last created file. To prevent deleting older files, set the maximum number of trace files to a large number. *Range:* `1` through `999` *Default:* `1`

Global Module Parameters

Set the following parameters to define configuration information that applies to the whole module (`module.#`). Set values for these parameters when you begin to define the configuration of a module (see *page 1298*).

Parameter	Value
set_api	Selects the ISDN layer that the Bfv API uses to configure layer configuration parameters that apply to all protocols supported by the module. Use `set_api=bfv` for all non-ISDN telephony protocols. When the module uses an ISDN protocol, set this parameter to either value. `BFV` Causes the Bfv API to configure parameters for layer 2 and layer 3 in addition to configuring layer 1 parameters. `BSMI` Causes the Bfv API to configure parameters only for layer 1. When you specify `BSMI` as the value for *set_api*, your application must configure the parameters for layer 2 and layer 3; otherwise, the Bfv API ignores these parameters. *Default:* `BFV`
auto_connect	Specifies a Boolean value that determines whether the call control system automatically connects the logical channels and the B-channels on the trunks. Set this parameter to: `FALSE` Does not make connections. When you specify this value, the application must make the connections for those Dialogic® Brooktrout® modules where you can make connections using software. `TRUE` Automatically connects DSP channels and ports. *Default:* `TRUE`

Parameter	Value

connections	Specifies a filename that contains a list of connections defining the relationship between the source and destination ports. This feature is not supported on analog and BRI boards, except those which also support H.100 (mezzanine board required).
	
	If you specify a filename as the value for this parameter, create the file using the following format:
	
	`[`
	`connect` `conn_mode src_port_class src_unit src_stream src_slot dest_port_class ...`
	`connect` `conn_mode src_port_class src_unit src_stream src_slot dest_port_class ...`
	`...`
	`]`
	
	Each of these lines begins with the keyword `connect`, and is followed in order by the connection mode, source port class, unit, stream, and slot, and the destination port class, unit, stream, and slot. The meanings of these values follow.
	
	If you use this parameter to specify connections, the call control ignores any value set in the *auto_connect* parameter.
	
	`conn_mode`
	
	Specifies the relationship between the source (S) and destination (D) ports with one of the following values. The value 7 is normally used.
	
	`1` Transmit only
	`2` Receive only
	`3` Full duplex
	`7` Full duplex with signaling
	`9` Transmit only, SD inverted
	`A` Receive only, SD inverted
	`B` Full duplex, SD inverted
	`F` Full duplex with signaling, SD inverted
	
	`src_port_class`
	
	Specifies the source telephony resource port class. Specify the value in hexadecimal.
	
	`0` Channel
	`E` Bus (for example, H.100)
	`F` Network (T1 or E1)

Parameter	Value

connections (continued)	`src_port_unit`
	Specifies the source resource port unit number for the source port class. If the class is 0 (channel), then the port unit value indicates the logical channel number. (When using *Bfv Line Attach*, the Bfv API sets the ordinal channel value to n – 2 where n is the logical channel number.)
	`src_stream`
	Specifies the source resource stream number.
	`src_slot`
	Specifies the source resource time slot number.
	`dest_port_class`
	Specifies the destination telephony resource port class. Specify this value in hexadecimal.
	`0` Channel
	`E` Bus (for example, H.100)
	`F` Network (T1 or E1)
	`dest_port_unit`
	Specifies the destination resource port unit number for the destination port class. If the class is 0 (channel), then the port unit value indicates the logical channel number. (When using *BfvLineAttach*, the Bfv API sets the ordinal channel value to n – 2 where n is the logical channel number.)
	`dest_stream`
	Specifies the destination resource stream number.
	`dest_slot`
	Specifies the destination resource time slot number.
	Default: No filename specified; the Bfv API does not make any connections.

Parameter	Value
pcm_law	Selects the PCM coding of the port. Call control also uses the value set for this parameter when establishing calls on protocols such as ISDN that provide layer 3 signaling using the pcm law. `ALAW` Configures the module to use A-law PCM on the ports and CTbus. `MULAW` Configures the module to use Mu-law PCM on the ports and CTbus. *Default:* Depends on the configuration of the first port on the module. `ALAW` if the port configuration is E1 or BRI. `MULAW` if the port configuration is T1 or Analog or Inactive.
jate_redial_ restriction	Specifies a value to configure for Japan's JATE redial restriction that prevents too many redials to the same called number. The Bfv API automatically activates this parameter for calls when the user sets the parameter `country_code` to `JAPAN` in the *btcall.cfg* user-defined configuration file (see *page 1155*). The Bfv API defines an unsuccessful attempt as a call that does not receive a CONNECT within 50 seconds or receives a DISCONNECT with a cause code indicating that the called party is busy. `none` Specifies that the number can be redialed any number of times. `15times` Specifies that the B-channel cannot make more than 15 consecutive unsuccessful calls to the same called party. `3in3mins` Specifies that the B-channel only allow 3 rejected calls to occur within 3 minutes. The 3-minute timer starts after the first unsuccessful attempt to connect to a called party number. After the timer expires, the application can redial the same called party with the restriction applied again. *Default:* `3in3mins`

Parameter	Value
jate_emergency_ numberN	Specifies a value that allows the application to supply one of a series of emergency numbers that can be redialed without restriction when the application applies the JATE redial restriction. The parameter allows you to specify up to 10 (ECC_MAX_EMERG_NUMS) different emergency numbers as: `jate_emergency_numberN = XXXX` where `N` indicates the series value from 0 through 9 and `XXXX` supplies the emergency telephone number to associate with the *N* value. When the module includes a redial restriction and the called party matches one of the numbered values, the module does not restrict the call. If you do not specify a called party number, the module redials all calls according to the specified restriction. If you define less than 10 numbers, use the lower numbers first (for example: `jate_emergency_number0`, `jate_emergency_number1`, `jate_emergency_number2`, and so on). Set this parameter as follows: *Range:* 1 through 255 character string (`ECC_MAX_DIGIT_STR` - 1) to specify the associated called party number. Specifies the called party number assigned to the *jate_emergency_numberN* series number where N is a value from 0 through 9. Use the lower numbers first for the series. *Default:* Blank

Parameter	Value
channels	Specifies the number of channels on either a hardware or virtual module configured to receive a firmware download.

Note: This parameter only applies when using the Boston Host Service (bostservice). If you use the service, you must start it before you start any applications (see your installation and configuration guide for instructions).

When the firmware is downloaded to a module for the first time, the assigned ordinal channel numbers start wherever the assignment left off on the previous module. As the system initializes the modules, this numbering process creates a continuous ordering of the channel assignments across all the modules in the system. On later downloads, each module's ordinals begin at the same location, regardless of any decrease or increase in the channel count of a lower-numbered module. Therefore, if you decrease the channel count for a lower numbered module, the process creates gaps in the channel numbering assignments, possibly affecting your application. If you attempt to increase the channel count above any module's initial channel count, the system ignores the added channels.

For the following situations, restart the driver whenever you want to:

1. Get a continuous assignment of channel numbers after decreasing the channel count on any module.
2. Increase the number of channels above a module's initial channel count.

Note: If you installed the Boston driver as a Plug and Play driver, you must reboot the system to restart the driver.

Set this parameter as follows:

`0`	Specifies downloading the firmware to the default value of the number of channels on the module.
`1 – 1024`	Specifies a value defining the number of channels on the module configured to receive a firmware download.
Range:	`1 – 1024` (not to exceed the maximum number of available channels on the module).
Value Type:	decimal
Default:	`0`

Parameter	Value
vb_firm	Indicates that the module is a virtual module and specifies the filename of the shared library that contains the loadable firmware for the virtual module. **Note:** This parameter only applies when using the Boston Host Service (bostservice). If you use the service, you must start it before you start any applications (see your installation and configuration guide for instructions). *Default:* No default. Absence of the parameter indicates that the module is not a virtual module.
static_ring_detect_enable	Specifies whether to turn static ring detection mode on or off for a module's inbound calls. When you turn on static ring detection mode on a module's configuration, your application must explicitly turn on the ring detection behavior on each call control channel that your application uses to receive inbound calls (see *Volume 2, BfvCallRingDetect* function). Set this parameter to: `FALSE` Turns static ring detection mode off for the module's inbound calls. `TRUE` Turns static ring detection mode on for the module's inbound calls. *Default:* `TRUE`

Unchangeable Internal Parameters

This section lists parameters that the user might find in the configuration file when manually editing the file.

Do not modify or delete any of these parameters when manually editing the call control configuration file.

The configuration tool adds these parameters to the configuration file. The presence or absence of any of these parameters only affects the configuration tool.

Parameter	Value
model	Indicates a value that identifies the name of a module. The configuration tool uses the value in this parameter as the cached information that identifies the module when in offline mode. Example: `TR1034+P4V4FH-E1-N`
exists	Indicates the state of a module. Set this parameter to: `0` Module does not exist `1` Module exists
cc_type	Indicates the type of call control the module is configured to use. Set this parameter to: `0` PSTN call control `1` IP call control
virtual	If present in the file, this parameter indicates that the module is a virtual module. When the parameter is absent, configuration applies to a hardware module.

Clock Configuration Parameters

In the `module.#/clock_config` section of the call control configuration file, set one or both of the following parameters to configure the clock for the module.

Parameter	Value
clock_mode	Specifies a value that determines whether the module drives the clock on the CT bus or receives its clocking from the CT bus. Set this parameter to: `MASTER` Configures the module to drive the clock on the CT bus. `SLAVE` Configures the module to receive clocking from the CT bus. *Default:* `MASTER`
clock_source	Specifies the source of the clock used to drive the CT bus. Set this parameter only if you set the value for *clock_mode* to `master`. The module derives its clock from: `Internal` The internal oscillator. `TrunkA` The network trunk, port A. `TrunkB` The network trunk, port B. `TrunkC` The network trunk, port C. `TrunkD` The network trunk, port D. `TrunkE` The network trunk, port E. `TrunkF` The network trunk, port F. `TrunkG` The network trunk, port G. `TrunkH` The network trunk, port H. `Netref1` The H.100/H.110 network reference (1) clock. `Netref2` The H.100/H.110 network reference (2) clock. `clock_a` The H.100/H.110 A clock. `clock_b` The H.100/H.110 B clock. *Default:* `TrunkA` **Note:** If you configure a port as `inactive` and inadvertently select it as the *clock_source*, the system cannot operate.

clock_compatibility

Specifies a value that determines how the module controls the MVIP-compatible clock signal. Set this parameter to:

`NONE` Does not configure the module to control compatible clocking.

`MVIP` For a module configured as the primary clocking source, setting this value configures the module to generate an MVIP clock signal.

For a module configured as the secondary clocking source, setting this value configures the module to receive its clock signal from the MVIP bus.

Default: `NONE`

Advanced H.100 and H.110 Parameters

Set the following parameters for a module configured to use an H.100 or H.110 bus connection.

Parameter	Value
bus_speed	Defines the speed of the H.100 bus when set to one of the following values: `2` Sets the bus speed to 2 MHz. `4` Sets the bus speed to 4 MHz. `8` Sets the bus speed to 8 MHz. *Default:* `2` (2 MHz)
master_drive	Specifies the H.100 clock that the primary module drives. Set this parameter to one of the following values only if you set the value in the *clock_mode* parameter to `MASTER`. `Clock_A` Configures the primary module to drive the A clock on the H.100 bus. `Clock_B` Configures the primary module to drive the B clock on the H.100 bus. *Default:* `Clock_A`
slave_source	Specifies the H.100 clocks that the secondary module uses. Set this parameter to one of the following values only if you set the value in the *clock_mode* parameter to `SLAVE`. `Clock_A` Configures the secondary module to source its clocking from clock A on the H.100 bus and enables clock B for auto-fallback if clock A fails. `Clock_B` Configures the secondary module to source its clocking from clock B on the H.100 bus and enables clock A for auto-fallback if clock B fails. `Clock_A_Nofallback` Configures the secondary module to source its clocking from clock A on the H.100 bus without enabling auto-fallback if clock A fails. `Clock_B_nofallback` Configures the secondary module to source its clocking from clock B on the H.100 bus without enabling auto-fallback if clock B fails. *Default:* `Clock_A`

Parameter	Value
master_ref_fallback	Specifies the source of the fallback clock used to drive the CT bus if the value specified for the *clock_source* parameter should fail. Set this parameter only if you set the value for *clock_mode* to `master`.

`Internal`	The module derives its fallback clock from the internal oscillator.
`TrunkA`	The module derives its fallback clock from network trunk, port A.
`TrunkB`	The module derives its fallback clock from network trunk, port B.
`TrunkC`	The module derives its fallback clock from network trunk, port C.
`TrunkD`	The module derives its fallback clock from network trunk, port D.
`TrunkE`	The module derives its fallback clock from network trunk, port E.
`TrunkF`	The module derives its fallback clock from network trunk, port F.
`TrunkG`	The module derives its fallback clock from network trunk, port G.
`TrunkH`	The module derives its fallback clock from network trunk, port H.
`Netref1`	The module derives its fallback clock from the H.100/H.110 network reference (1) clock.
`Netref2`	The module derives its fallback clock from the H.100/H.110 network reference (2) clock.
`clock_a`	The module derives its fallback clock from the H.100/H.110 A clock.
`clock_b`	The module derives its fallback clock from the H.100/H.110 B clock.
`Disabled`	The module does not have a fallback clock defined. Using this value prevents a fallback from occurring.

Default: `Disabled`

Note: If you configure a port as `inactive` and inadvertently select it as the *clock_source*, the system cannot operate.

The Bfv API automatically sets the fallback clock to `Disabled` if the system does not have a Trunk B.

Port Configuration Parameters

Set the following parameters when you begin to define the configuration of a port.

The Bfv API supports specific configuration parameters for the following protocols and line interfaces that the port might use:

- Analog Direct Inward Dialing (DID) (see *page 1194*)
- Analog Loop Start (see *page 1197*)
- ISDN Basic Rate Interface (BRI) (see *page 1201*)
- E1 ISDN Primary Rate Interface (PRI) (see *page 1209*)
- E1 CAS (see *page 1217*)
- E1 CAS R2 (see *page 1221*)
- E1 and T1 QSIG (see *page 1223*)
- T1 ISDN Primary Rate Interface (PRI) (see *page 1232*)
- T1 Robbed Bit Signaling (RBS) (see *page 1240*)

➢ *To define the configuration of a port:*

1. Set a value for the *port_config* parameter.
2. Set values for the remaining port configuration parameters as applicable.
3. Set the configuration-specific parameters for the port depending on the value specified for the *port_config* parameter.

Parameter	Value
port_config	Specifies one of the following values that defines the CT bus or line type to configure for the port.

`INACTIVE`	Disabled port
`ANALOG`	Analog Loop Start line
`ANALOG_DID`	Analog Direct Inward Dialing (DID) line
`BRI`	Basic Rate Interface
`E1_ISDN`	E1 ISDN
`E1_CAS`	E1 CAS
`E1_R2_CAS`	E1 CAS R2
`E1_DPNSS`	E1 DPNSS (not supported in this release)
`E1_QSIG`	E1 QSIG
`T1_QSIG`	T1 QSIG
`T1_ISDN`	T1 ISDN
`T1_ROBBED_BIT`	T1 RBS

Default: `INACTIVE`

Note: Any port defined as `INACTIVE` completes configuration requirements for the port. The configuration-specific parameters that start on *page 1193* do not apply to inactive or disabled ports.

fractional_start_channel

Specifies the zero-based index value of the first channel available on the port.

If you do not specify this parameter, the Bfv API assumes that the full port is available.

When you configure a fractional port, the *auto_connect* (see *page 1178*) feature only connects the channels in use.

The valid values are:

`0 – 29`	E1 (all types)
`0 – 22`	T1 ISDN
`0 – 23`	T1 RBS

Range: `0 – 29`

Default: `0`

Parameter	Value
fractional_channel_count	Specifies a value that indicates the number of channels available on the fractional port. Use this parameter in conjunction with the *fractional_start_channel* parameter. **Note:** Set this parameter to –1 when not using fractional channels. Setting this parameter to 0 makes zero B-channels available. Valid values are: `–1` Indicates that a fractional port is not in use. `0 – 29` Indicates the number of available E1 channels on the fractional port. `0 – 22` Indicates the number of available T1 ISDN channels on the fractional port. `0 – 23` Indicates the number of available T1 RBS channels on the fractional port. *Range:* `–1, 0 – 29` *Default:* `–1`
wait_for_service_timeout	Specifies a value that indicates the amount of time to wait for a trunk to come into service before returning an error when placing an outbound call. If this timeout expires, the outbound call fails. If the trunk comes into service before this timeout expires, the system places the call normally. *Unit:* second *Range:* `0 – 10,000` *Default:* `10`

Specific Parameters for Port Configuration

After setting one or more parameters to configure the port, proceed to one of the following pages to set the protocol variant or interface parameters for the port. For each value you set for the *port_config* parameter, you must continue to configure the port by setting the configuration parameters that specifically relate to the selected *port_config* as follows:

- *Analog Direct Inward Dialing (DID) — see page 1194*
- *Analog Loop Start — see page 1197*
- *ISDN BRI — see page 1201*
- *E1 ISDN — see page 1209*
- *E1 CAS — see page 1217*
- *E1 CAS R2 — see page 1221*
- *E1 and T1 QSIG — see page 1223*
- *T1 ISDN — see page 1232*
- *T1 RBS — see page 1240*

Configuration Parameters for Analog Direct Inward Dialing (DID) Ports

Set one or more of the following parameters to configure a port that uses an analog DID protocol (see *page 1306* for an example configuration file).

Parameter	Value
did_offset	Specifies a value that defines the number of digits to remove from the beginning of the string of DID digits (see *max_did_digits on page 1242*) received from the network. If max_did_digits is set to 0, or if the number of DID digits received is less than the number specified by max_did_digits, this parameter has no effect. Set this parameter to: `0` Does not remove any DID digits. `1 – 255` Specifies the number of digits to remove. *Range:* `0` through `255` *Default:* `0`
did_timeout	Specifies a value that defines the maximum timeout allowed before processing the call after assuming receipt of the last DID digit. Set this parameter to: `0` Indicates no waiting time. `1 – 20` Specifies the number of seconds to allow after receiving the last DID digit before processing the call. *Unit:* second *Range:* `0` through `20` *Default:* `5` (used when the Bfv API does not find another value for this parameter).
input_gain	Specifies a value that defines the number of decibels (dB) to increase or decrease the power of the incoming audio signal on the phone line. Set this parameter to: `+6` Increases the amount of gain by +6 dB. `0` Does not increase or decrease the amount of gain. `–6` Decreases the amount of gain by –6 dB. *Unit:* 1 dB *Range:* `+6` through `–6` *Default:* `0`

Parameter	Value
line_coef	Specifies one of the following values that defines the impedance and other electrical characteristics of the line: `0` Specifies an impedance of 600 ohms (Ω). `1` Specifies an impedance of 900 Ω. `2` Specifies impedance and electrical characteristics conforming to the CTR21 standard (270 Ω + 750 Ω ‖ 150 nF). `3` Specifies impedance and electrical characteristics conforming to standards for Australia and New Zealand (220 Ω + 820 Ω ‖ 120 nF). `4` Specifies impedance and electrical characteristics conforming to standards for Slovakia, Slovenia, and South Africa (220 Ω + 820 Ω ‖ 115 nF). `5` Specifies impedance and electrical characteristics conforming to standards for China (200 Ω + 600 Ω ‖ 100 nF). *Default:* `0` (600 ohms)
max_did_digits	Specifies a value that defines the maximum number of DID digits to expect before accepting an incoming call. Set this parameter to: `0` Turns off waiting for DID digits. `1 – 255` Specifies the number of digits. *Range:* `0` through `255` *Default:* `0` **Note:** The system only reports the expected number of DID digits (the value specified for *max_did_digits*) to the application even if the number of DID digits it received from the network exceeds the number specified for *max_did_digits*.
output_gain	Specifies a value that defines the number of decibels (dB) to increase or decrease the power of the outgoing audio signal on the phone line. Set this parameter to: `+6` Increases the amount of gain by +6 dB. `0` Does not increase or decrease the amount of gain. `–6` Decreases the amount of gain by –6 dB. *Unit:* 1 dB *Range:* `+6` through `–6` *Default:* `0`

Parameter	Value

protocol_file	Specifies the name of the analog DID protocol file to load for the port. This parameter dictates the protocol that runs on the port. Set this parameter to:
	`immediatedial.lec`
	`winkstart.lec`
	Default: `winkstart.lec`

reject_incomplete_did	Specifies the action to take when the number of DID digits received from the incoming call is less than the number of digits specified for the *max_did_digits* parameter. Set this parameter to: `FALSE` Reports the call to the application even if the number of received DID digits is less than the *max_did_digits* value. The system takes this action when the number of digits collected remains incomplete after the *did_timeout* period. `TRUE` Automatically sends the network a reject message that causes the network to drop the call. The application does not receive any notification of the call, but the system plays a fast busy tone to the caller. *Default:* `FALSE`

Other Parameters

In addition to the configuration-specific parameters for an analog DID port, the Bfv API also uses parameters from the *BT_CPARM.CFG* file to configure lines. See *BT_CPARM.CFG Parameter File* on *page 1441* for descriptions and values for the following parameters.

- *loopcur_debounce*
- *loopcur_len*
- *min_on_hook*
- *pre_wink*
- *post_wink*

Configuration Parameters for Analog Loop Start Ports

Set one or more of the following parameters to configure an analog loop start port (see *page 1298* for an example configuration file).

Parameter	Value
caller_id	Specifies a value that indicates whether detection of V.23-based caller ID has been enabled. Set the parameter as follows: `DISABLED` Turns off detection of V.23-based caller ID. `ENABLED` Turns on detection of V.23-based caller ID. **Note:** In North America and some other locations, the system sends the caller ID signal between the first and second rings. To detect the caller ID correctly, you must set the *num_rings* (*page 1199*) parameter to a value of 2 or greater to prevent the system from reporting the call to the application before the caller ID has been sent by the Central Office. *Default:* `ENABLED`
country	Specifies the name of the file that matches a binary file containing the country coefficients to use for the port. *Range:* Maximum of 256 characters (`_MAX_PATH`) *Default:* None
did_offset	Specifies a value that defines the number of digits to remove from the beginning of the string of DID digits (see *max_did_digits on page 1242*) received from the network. If max_did_digits is set to 0, or if the number of DID digits received is less than the number specified by max_did_digits, this parameter has no effect. Set this parameter to: `0` Does not remove any DID digits. `1 – 255` Specifies the number of digits to remove. *Range:* `0` through `255` *Default:* `0`
flash_hook_duration	Specifies a value that defines the duration of a flash hook signal. This parameter defines the amount of time to place the line on hook during a flash hook. *Unit:* 10 ms *Range:* `1 – 500` *Default:* `50`

Parameter	Value
input_gain	Specifies a value that defines the number of decibels (dB) to increase or decrease the power of the incoming audio signal on the phone line. Set this parameter to: `+6` Increases the amount of gain by +6 dB. `0` Does not increase or decrease the amount of gain. `–6` Decreases the amount of gain by –6 dB. *Unit:* 1 dB *Range:* `+6` through `–6` *Default:* `0`
loop_reversal_ for_connect	Specifies how to interpret a loop reversal signal as an audio path connection indication. Set this parameter to: `DISABLED` Ignores loop reversal as an indication of audio path connection. `ENABLED` Interprets loop reversal as an indication of audio path connection. *Default:* `DISABLED`
loop_reversal_ for_disconnect	Specifies how to interpret a loop reversal signal as a call disconnect indication. Set this parameter to: `DISABLED` Ignores loop reversal as a call disconnect signal. `ENABLED` Interprets loop reversal as a call disconnect signal. *Default:* `DISABLED`
max_did_digits	Specifies a value that defines the maximum number of DID digits to expect before accepting an incoming call. This parameter is only valid when you set the *country_code* parameter to `JAPAN` in the user-defined configuration file (see *page 1155*). Set this parameter to: `0` Disables waiting for DID digits. `1 – 4` Specifies the number of digits to expect before accepting an incoming call. *Range:* `0` through `4` *Default:* `0`

Parameter	Value
missing_wait	Specifies a value that defines the amount of time that the system must pause before dialing after it detects a missing dial initiation character. (In the GUI tool for Windows users, this parameter is called *Dial Initiation Character Timeout*.) Dial initiation characters are: ‘,’ (comma) Causes a one-second pause. ‘w’ Waits for dial tone. ‘;’ (semicolon) Causes a five-second pause. ‘i’ or ‘I’ Causes a five-second pause. *Unit:* 10 ms *Range:* `0` through `1000` *Default:* `100` (1 second)
num_rings	Specifies a value that defines the number of rings the system must detect before the system reports a new incoming call to the application. *Range:* `1` to `255` *Default:* `2`
output_gain	Specifies a value that defines the number of decibels (dB) to increase or decrease the power of the outgoing audio signal on the phone line. Set this parameter to: `+6` Increases the amount of gain by +6 dB. `0` Does not increase or decrease the amount of gain. `–6` Decreases the amount of gain by –6 dB. *Unit:* 1 dB *Range:* `+6` through `–6` *Default:* `0`

protocol_file	Specifies the full path and name of the protocol file to load for the analog port. Most of the time a path should be used for this file name. Values include:
	`analog_loopstart_europe.lec`
	`analog_loopstart_us.lec`
	Makes calls using the specified loop start protocol.
	Default: `analog_loopstart_us.lec`

Parameter	Value
transfer_variant	Specifies the transfer method that the network (refer to the vendor specifications for your switch) runs for call transfers or disables call transfer. Set this parameter to: `NONE` Disables call transfer. `HOOKFLASH` Specifies a hook flash transfer. *Default:* `HOOKFLASH` The SR140 does not support call transfer.

Other Parameters

In addition to the configuration-specific parameters for an analog port, the Bfv API also uses parameters from the *BT_CPARM.CFG* file to configure analog lines. See *BT_CPARM.CFG Parameter File* on *page 1441* for descriptions and values for the following parameters.

- *dial_tone_min*
- *dtone_timeout_highbyte*
- *loopcur_timeout*
- *loop_max_break*
- *min_on_hook*
- *pulse_inter_time*
- *pulse_max_break*
- *ring_blank*
- *tone_inter_time*
- *dtone_len*
- *dtone_timeout_lowbyte*
- *loop_seizure*
- *max_interdigit*
- *pulse_break*
- *pulse_make*
- *pulse_min_break*
- *ring_len*
- *tone_len*

Configuration Parameters for BRI Ports

Set one or more of the following parameters to configure a BRI port (see *page 1300* for an example configuration file).

Note: The CO emulation (*emulation*) is only used during a test environment. The BRI emulation, in particular, is only used on Port A of the BRI board.

Parameter	Value
call_type	Specifies the call type to use when making the outbound call. Use one of the following values for this parameter: `AUTO` Makes a call using the modem type and then automatically retries the call using the voice type if the other end cannot accept modem calls. `MODEM` Makes a modem (3.1 kHz audio) call. This setting provides higher quality audio for the call. `SPEECH` Makes a voice call. *Default:* `AUTO`
datalink	Defines whether to configure the port's call switch for a point-to-point or point-to-multipoint circuit. Set this parameter to: `AUTO` Automatically detects the datalink setting. `POINT2POINT` Configures for a point-to-point circuit. `POINT2MULTIPOINT` Configures for a point-to-multipoint circuit. *Default:* `AUTO`
default_caller_id_channel_0	Specifies a string of up to 15 characters that provides the caller ID to use when placing outbound calls on channel 0. If the user application provides a caller ID when placing a call, the system ignores this parameter. *Unit:* character string *Range:* `1 – 15` (`_ECC_MAX_ANI_LENGTH`) *Default:* <blank>
default_caller_id_channel_1	Specifies a string of up to 15 characters that provides the caller ID to use when placing outbound calls on channel 1. If the user application provides a caller ID when placing a call, the system ignores this parameter. *Unit:* character string *Range:* `1 – 15` (`_ECC_MAX_ANI_LENGTH`) *Default:* <blank>

Parameter	Value
did_offset	Specifies a value that defines the number of digits to remove from the beginning of the string of DID digits (see *max_did_digits on page 1203*) received from the network. If max_did_digits is set to 0, or if the number of DID digits received is less than the number specified by max_did_digits, this parameter has no effect. Set this parameter to: `0` Does not remove any DID digits. `1 – 255` Specifies the number of digits to remove. *Range:* `0` through `255` *Default:* `0`
did_timeout	Specifies a value that defines the maximum timeout allowed before processing the call after assuming receipt of the last DID digit. Set this parameter to: `0` Indicates no waiting time. `1 – 20` Specifies the number of seconds to allow after receiving the last DID digit before processing the call. *Unit:* second *Range:* `0` through `20` *Default:* `10` (used when the Bfv API does not find another value for this parameter)
disable_call_proceed	Specifies a value that determines whether the system sends a CALL PROCEEDING indication after receiving a SETUP message from the network. Set this parameter to: `FALSE` Indicates that the system sends a CALL PROCEEDING message after receiving a SETUP message from the network. `TRUE` Indicates that the system does not send a CALL PROCEEDING message after receiving a SETUP message from the network. *Default:* `FALSE`
disable_conn_ack	Specifies whether the system sends a connection acknowledgment after receiving a connect message from the network. Set this parameter to: `FALSE` The system sends a connection acknowledgment message after receiving a connect message from the network. `TRUE` The system does not send a connection acknowledgment message after receiving a connect message from the network. *Default:* `FALSE`

Parameter	Value
emulation	Specifies whether to configure the trunk for Central Office (CO) or Customer Premise Equipment (CPE) protocol emulation. Set this parameter for testing purposes only. `CO` Emulates the CO protocol. `CPE` Emulates the CPE protocol. *Default:* `CPE`
max_did_digits	Specifies a value that defines the maximum number of DID digits to expect before accepting an incoming call. Set this parameter to: `0` Disables waiting for DID digits. `1 – 255` Specifies the number of digits to expect before accepting an incoming call. *Range:* `0` through `255` *Default:* `0` **Note:** The system can report all of the DID digits it received from the network to the application even if the number of received DID digits exceeds the number specified for *max_did_digits*. To remove the excess digits, set the *did_offset* parameter (see *page 1202*) so that the system only passes the expected number of digits to the application.
max_overlapped_digits	Specifies the maximum number of digits to send when the application supports overlapped dialing for longer phone numbers. Set this parameter to: `0` Disables support for overlapped dialing. *Range:* `1 – 24 (IISDN_MAX_DIGITS)` *Default:* `20`

Parameter	Value
msn_x	Specifies a value that allows the application to supply one of a series of multiple subscriber numbers (MSN) to acknowledge. The parameter allows you to specify up to 10 different MSN numbers as: `msn_x = XXXX` where `x` indicates the series value from 0 through 9 and `XXXX` supplies the multiple subscriber telephone number to associate with the `x` value. When you set one or more of the ten numbers and the port operates in point-to-multipoint mode, the port only acknowledges incoming calls to a called party that matches one of the numbered values. If you do not specify a value for this parameter, the port answers all calls presented to it. If you define less than 10 numbers, use the lower numbers first (for example: `msn_0`, `msn_1`, `msn_2`, and so on). Valid values are: *Unit:* character string *Range:* 1 through 15 (`_ECC_MAX_ANI_LENGTH`) to specify the associated called party number. Specifies the called party number assigned to the *msn_x* series number where *x* is a value from 0 through 9. Use the lower numbers first for the *msn_x* series. *Default:* <blank>
numbering_plan	Specifies a value that identifies the type of numbering plan used for outbound calls (called party number). Set this parameter to: `ISDN` Indicates that the port uses an ISDN numbering plan. `PRIVATE` Indicates that the port uses a private numbering plan. `TELEPHONY` Indicates that the port uses a telephony numbering plan. `UNKNOWN` Indicates that the port uses an unknown numbering plan. *Default:* `UNKNOWN`

Parameter	Value
numbering_type	Specifies a value that identifies the type of telephone number used for outbound calls (called party number). Set this parameter to: `ABBREVIATED` Indicates that the port uses an abbreviated numbering type. `INTERNATIONAL` Indicates that the port uses an international numbering type. `NATIONAL` Indicates that the port uses a national (North American) numbering type. `SUBSCRIBER` Indicates that the port uses a subscriber numbering type. `UNKNOWN` Indicates that the port uses an unknown numbering type. *Default:* `UNKNOWN`
preferred	Specifies a value that causes outbound calls to set a use preference for the B channel when the port uses a point-to-multipoint circuit. Set this parameter only if the port uses a point-to-multipoint circuit. If other devices share the BRI line with the Dialogic® Brooktrout® hardware, you must set this parameter to `TRUE` to make the B channel setting preferred on outbound calls. If you set the parameter to `FALSE`, making the B channel setting exclusive for an outbound call, the port's point-to-multipoint circuit cannot operate. Set this parameter to: `FALSE` Sets the B channel to *exclusive* on outbound calls. `TRUE` Sets the B channel to *preferred* on outbound calls. *Default:* `TRUE`
presentation	Specifies a value that indicates the type of presentation of the calling party number the port uses when placing an outbound call. Set this parameter to: `ALLOWED` Indicates that the port allows presentation of the calling party number to the called party. `NUM_NOT_AVAIL` Indicates that the port does not have a calling party number specified to present to the called party. `RESTRICTED` Indicates that the port restricts presentation of the calling party number to specific called party numbers. APP_DEFINED Indicates that the application passes in the value for this parameter. **Note:** You should set the parameter to `NUM_NOT_AVAIL` when connected to a public network. *Default:* `ALLOWED`

Parameter	Value
redirect_as_calling_party	Specifies a value that selects an option to use the redirect number as the calling party number reported to the application. Set this parameter to: `FALSE` Causes the system to use the original calling party number as the number reported to the application. `TRUE` Causes the system to use the redirect number as the calling party number reported to the application. Selecting this option removes any association between the original calling party number and the call. *Default:* `FALSE`
reject_incomplete_did	Specifies the action to take when the number of DID digits received from the incoming call is less than the number of digits specified for the *max_did_digits* parameter. Set this parameter to: `FALSE` Sends an alerting or proceeding message to the network and reports the call to the application even if the number of received DID digits is less than the *max_did_digits* value. The system takes this action when the number of digits collected remains incomplete after the *did_timeout* period or when it receives a sending complete informational element (IE). `TRUE` Sends the network a reject message that causes the network to drop the call. The application does not receive any notification of the call. *Default:* `FALSE`

Parameter	Value

screening	Specifies a value that indicates whether the port provides and validates the calling party number passed to the called party. Set this parameter to:
	`NETWORK_PROVIDED`
	Indicates that the network validates the calling party number.
	`USER_NOT_SCREENED`
	Indicates that the port provides the calling party number without validating it.
	`USER_VERIFICATION_FAILED`
	Indicates that the port failed to validate the calling party number.
	`USER_VERIFICATION_PASSED`
	Indicates that the port provided the calling party number and passed a successfully validated number to the called party.
	`APP_DEFINED`
	Indicates that the application passes in the value for this parameter.
	Note: Set the parameter to `USER_NOT_SCREENED` when connected to a public network.
	Default: `USER_NOT_SCREENED`

send_dialcomplete	Specifies whether the system sends an informational element (IE) for outbound calls that indicates the end of dialing. Set this parameter to: `FALSE` Indicates that the port does not transmit a DIAL COMPLETE message on outbound calls. `TRUE` Requests the system to transmit a DIAL COMPLETE message on outbound calls. *Default:* `TRUE`
spid	Specifies a value that indicates a number assigned as a service profile identifier (SPID). The USA sometimes uses this identifier but European nations do not. Leave the parameter blank unless the service requires an identifier. *Unit:* character *Range:* `1 – 15` (`_ECC_MAX_ANI_LENGTH`) *Default:* <blank>

Parameter	Value
transfer_variant	Specifies the transfer method that the network (refer to the vendor specifications for your switch) runs for call transfers, or disables call transfer. The SR140 does not support call transfer. Set this parameter to: `NONE` Disables call transfer. `ETSI_EXP_LINK` Specifies an ETSI transfer with explicit linkage. `ETSI_IMP_LINK` Specifies an ETSI transfer with implicit linkage. If the Bfv API cannot perform implicit linkage because there are more than two calls on the D-channel, the Bfv API performs explicit linkage. `NTT` Specifies a JATE active redirecting transfer. `NTT_MP` Specifies a JATE active redirecting transfer for a point to multipoint configuration. *Default:* `ETSI_IMP_LINK` **Note:** `NTT` is the default value when the user sets the parameter `country_code` to `JAPAN` in the *btcall.cfg* user-defined configuration file (see *page 1155*). `NTT_MP` is the default value when the user sets the `country_code` value to `JAPAN` and the `datalink` parameter value to `POINT2MULTIPOINT` (see *page 1201*).
wait_for_conn_ack	Specifies whether the system waits for the network to acknowledge a connect request before notifying the application that a call has been answered. Set this parameter to: `FALSE` Specifies that the system does not wait for the network to acknowledge a connect request before advancing an incoming call to the connected state. `TRUE` Requests the system to wait for the network to acknowledge a connect request before advancing an incoming call to the connected state. *Default:* `FALSE`

Configuration Parameters for E1 ISDN Ports

Set one or more of the following parameters to configure an E1 ISDN port (see *page 1301* for an example configuration file).

Parameter	Value
call_type	Specifies the call type to use when making the outbound call. Use one of the following values for this parameter: `AUTO` Makes a call using the modem type and then automatically retries the call using the voice type if the other end cannot accept modem calls. `MODEM` Makes a modem (3.1 kHz audio) call. This setting provides higher quality audio for the call. `SPEECH` Makes a voice call. *Default:* `AUTO`
crc	Specifies a value that indicates whether the port has cyclical redundancy checking (CRC) turned on. Set this parameter to: `DISABLED` Turns off CRC for the port. `ENABLED` Turns on CRC for the port. *Default:* `ENABLED`
default_caller_id	Specifies a string of up to 15 characters that provides the caller ID to use when placing outbound calls. If the user application provides a caller ID when placing a call, the system ignores this parameter. *Unit:* character string *Range:* `1 – 15` (`_ECC_MAX_ANI_LENGTH`) *Default:* Blank
did_offset	Specifies a value that defines the number of digits to remove from the beginning of the string of DID digits (see *max_did_digits on page 1211*) received from the network. If max_did_digits is set to 0, or if the number of DID digits received is less than the number specified by max_did_digits, this parameter has no effect. Set this parameter to: `0` Does not remove any DID digits. `1 – 255` Specifies the number of digits to remove. *Range:* `0` through `255` *Default:* `0`

Parameter	Value
did_timeout	Specifies a value that defines the maximum timeout allowed before processing the call after assuming receipt of the last DID digit. Set this parameter to: `0` Indicates no waiting time. `1 – 20` Specifies the number of seconds to allow after receiving the last DID digit before processing the call. *Unit:* second *Range:* `0` through `20` *Default:* `10` (used when the Bfv API does not find another value for this parameter)
disable_call_proceed	Specifies a value that determines whether the system sends a CALL PROCEEDING indication after receiving a SETUP message from the network. Set this parameter to: `FALSE` Indicates that the system sends a CALL PROCEEDING message after receiving a SETUP message from the network. `TRUE` Indicates that the system does not send a CALL PROCEEDING message after receiving a SETUP message from the network. *Default:* `FALSE`
disable_conn_ack	Specifies whether the system sends a connection acknowledgment after receiving a connect message from the network. Set this parameter to: `FALSE` The system sends a connection acknowledgment message after receiving a connect message from the network. `TRUE` The system does not send a connection acknowledgment message after receiving a connect message from the network. *Default:* `FALSE`
emulation	Specifies whether to configure the trunk for Central Office (CO) or Customer Premise Equipment (CPE) protocol emulation. Use this parameter for testing purposes only. Valid values include: `CO` Emulates the CO protocol. `CPE` Emulates the CPE protocol. *Default:* `CPE`
line_coding	Specifies a value defining the type of line encoding to use for the port. Set this parameter to: `AMI` Selects Alternate Mark Inversion. `HDB3` Selects High Density Bipolar Order 3. *Default:* `HDB3`

Parameter	Value
line_impedance	Specifies a value that defines the line impedance the port uses. Set this parameter to: `75` Specifies that the port uses an impedance value of 75 ohms. `120` Specifies that the port uses an impedance value of 120 ohms. *Default:* `120`
max_did_digits	Specifies a value that defines the maximum number of DID digits to expect before accepting an incoming call. Set this parameter to: `0` Disables waiting for DID digits. `1 – 255` Specifies the number of digits to expect before accepting an incoming call. *Range:* `0` through `255` *Default:* `0` **Note:** The system can report all of the DID digits it received from the network to the application even if the number of received DID digits exceeds the number specified for *max_did_digits*. To remove the excess digits, set the *did_offset* parameter (see *page 1209*) so that the system only passes the expected number of digits to the application.
max_overlapped_digits	Specifies the maximum number of digits to send when the application supports overlapped dialing for long phone numbers. Set this parameter to: `0` Disables support for overlapped dialing. *Range:* `1 – 24 (IISDN_MAX_DIGITS)` *Default:* `20`

numbering_plan	Specifies a value that identifies the type of numbering plan used for outbound calls (called party number). Set this parameter to:
	`ISDN`
	Indicates that the port uses an ISDN numbering plan.
	`PRIVATE`
	Indicates that the port uses a private numbering plan.
	`TELEPHONY`
	Indicates that the port uses a telephony numbering plan.
	`UNKNOWN`
	Indicates that the port uses an unknown numbering plan.
	Default: `UNKNOWN`

Parameter	Value

numbering_type	Specifies a value that identifies the type of telephone number used for outbound calls (called party number). Set this parameter to:
	`ABBREVIATED`
	Indicates that the port uses an abbreviated numbering type.
	`INTERNATIONAL`
	Indicates that the port uses an international numbering type.
	`NATIONAL`
	Indicates that the port uses a national (North American) numbering type.
	`SUBSCRIBER`
	Indicates that the port uses a subscriber numbering type.
	`UNKNOWN`
	Indicates that the port uses an unknown numbering type.
	Default: `UNKNOWN`

presentation	Specifies a value that indicates the type of presentation of the calling party number the port uses when placing an outbound call. Set this parameter to:
	`ALLOWED`
	Indicates that the port allows presentation of the calling party number to the called party.
	`NUM_NOT_AVAIL`
	Indicates that the port does not have a calling party number specified to present to the called party.
	`RESTRICTED`
	Indicates that the port restricts presentation of the calling party number to specific called party numbers.
	Note: You should set the parameter to `NUM_NOT_AVAIL` when connected to a public network.
	Default: `ALLOWED`

Parameter	Value

protocol	Specifies the type of protocol variant to use for the port. Set this parameter to:
	`EURO`
	NET-5 standard for PRI connections throughout Europe (also referred to as Euro-ISDN). Choosing this variant changes the layer 2 protocol timers to their appropriate NET-5 defaults.
	`1TR6`
	1TR6 standard for PRI connections in Germany.
	`VN3`
	VN3 standard for France.
	`Q931`
	General ITU-T Q.931 conformance.
	`Jate`
	INS-1500 for Japan.
	Default: `EURO`

reject_incomplete_did	Specifies the action to take when the number of DID digits received from the incoming call is less than the number of digits specified for the *max_did_digits* parameter. Set this parameter to: `FALSE` Sends an alerting or proceeding message to the network and reports the call to the application even if the number of received DID digits is less than the *max_did_digits* value. The system takes this action when the number of digits collected remains incomplete after the *did_timeout* period or when it receives a sending complete informational element (IE). `TRUE` Sends the network a reject message that causes the network to drop the call. The application does not receive any notification of the call. *Default:* `FALSE`
sabme	Specifies whether the port sends layer 2 Set Asynchronous Balanced Mode Extended (SABME) messages. Set this parameter to: `FALSE` Indicates that the port does not send SABME messages. `TRUE` Indicates that the port sends layer 2 SABME messages. *Default:* `TRUE` (default when `emulation = CPE`) `FALSE` (default when `emulation = CO`)

Parameter	Value

screening	Specifies a value that indicates whether the port provides and validates the calling party number passed to the called party. Set this parameter to:
	`NETWORK_PROVIDED`
	Indicates that the network validates the calling party number.
	`NONE`
	Indicates that the port does not provide a calling party number to the called party.
	`USER_NOT_SCREENED`
	Indicates that the port provides the calling party number without validating it.
	`USER_VERIFICATION_FAILED`
	Indicates that the port failed to validate the calling party number.
	`USER_VERIFICATION_PASSED`
	Indicates that the port provides the calling party number and passes a successfully validated number to the called party.
	Note: You should set the parameter to `USER_NOT_SCREENED` when connected to a public network.
	Default: `USER_NOT_SCREENED`

send_dialcomplete	Specifies whether the system sends an informational element (IE) for outbound calls that indicates the end of dialing. Set this parameter to: `FALSE` Indicates that the port does not transmit a DIAL COMPLETE message on outbound calls. `TRUE` Requests the system to transmit a DIAL COMPLETE message on outbound calls. *Default:* `TRUE`
send_restart	Specifies a Boolean value that determines whether the system sends a RESTART message after re-establishing layer 2. Set the field as follows: `FALSE` Indicates that the system does not send a RESTART message after re-establishing layer 2. `TRUE` Indicates that the system sends a RESTART message after re-establishing layer 2. *Default:* `FALSE`

Parameter	Value

switch_type	Specifies a value indicating the type of switch used for the board connection. Set this parameter to:
	`ATT_4ESS`
	AT&T #4 ESS.
	`ATT_5ESS`
	AT&T #5 ESS.
	`NTI_DMS100`
	Nortel DMS-100.
	`NTI_DMS250`
	Nortel DMS-250.
	`MD110_T1`
	Selects Ericsson MD-110 switch for North America.
	`MD110_E1`
	Selects Ericsson MD-110 switch (International).
	`SIEMENS`
	Siemens.
	`NTT`
	Japan.
	`UNKNOWN`
	Selects a switch type that complies with the ITU-T standards.
	Default: `UNKNOWN` (ITU-T compliant)

transfer_variant	Specifies the transfer method that the network (refer to the vendor specifications for your switch) runs for call transfers or disables call transfer. The SR140 does not support call transfer. Set this parameter to: `NONE` Disables call transfer. `ETSI_EXP_LINK` Specifies an ETSI transfer with explicit linkage. `NTT` Specifies a JATE active redirecting transfer. *Default:* `ETSI_EXP_LINK` **Note:** `NTT` is the default value when the user sets the parameter `country_code` to `JAPAN` in the *btcall.cfg* user-defined configuration file (see *page 1155*).

Parameter	Value
wait_for_conn_ack	Specifies whether the system waits for the network to acknowledge a connect request before notifying the application that a call has been answered. Set this parameter to: `FALSE` Specifies that the system does not wait for the network to acknowledge a connect request before advancing an incoming call to the connected state. `TRUE` Requests the system to wait for the network to acknowledge a connect request before advancing an incoming call to the connected state. *Default:* `FALSE`

Configuration Parameters for E1 CAS Ports

Set one or more of the following parameters to configure an E1 CAS port (see *page 1301* for an example configuration file).

Parameter	Value
caller_id	Specifies a value that indicates whether detection of V.23-based caller ID has been enabled. Set this parameter to: `DISABLED` Turns off detection of V.23-based caller ID. `ENABLED` Turns on detection of V.23-based caller ID. **Note:** In North America and some other locations, the system sends the caller ID signal between the first and second rings. To detect the caller ID correctly, you must set the *num_rings* (*page 1199*) parameter to a value of 2 or greater to prevent the system from reporting the call to the application before the caller ID has been sent by the Central Office. *Default:* `ENABLED`
crc	Specifies a value that indicates whether the port has cyclical redundancy checking (CRC) turned on. Set this parameter to: `DISABLED` Turns off CRC for the port. `ENABLED` Turns on CRC for the port. *Default:* `ENABLED`
did_offset	Specifies a value that defines the number of digits to remove from the beginning of the string of DID digits (see *max_did_digits on page 1219*) received from the network. If max_did_digits is set to 0, or if the number of DID digits received is less than the number specified by max_did_digits, this parameter has no effect. Set this parameter to: `0` Does not remove any DID digits. `1 – 255` Specifies the number of digits to remove. *Range:* 0 through 255 *Default:* `0`

Parameter	Value
did_timeout	Specifies a value that defines the maximum timeout allowed before processing the call after assuming receipt of the last DID digit. Set this parameter to: `0` Indicates no waiting time. `1 – 20` Specifies the number of seconds to allow after receiving the last DID digit before processing the call. *Unit:* second *Range:* `0` through `20` *Default:* `10` (used when the Bfv API does not find another value for this parameter).
flash_hook_duration	Specifies a value for the duration of a flash hook signal. This parameter defines the amount of time to place the line on hook (loop current dropped) during a flash hook. Set the value in units of 10 ms. *Unit:* 10 ms *Range:* `1 – 500` *Default:* `50`
line_coding	Specifies a value defining the type of line encoding to use for the port. Set this parameter to: `AMI` Selects Alternate Mark Inversion. `HDB3` Selects High Density Bipolar Order 3. *Default:* `HDB3`
line_impedance	Specifies a value that defines the line impedance the port uses. Set this parameter to: `75` Specifies that the port uses an impedance value of 75 ohms. `120` Specifies that the port uses an impedance value of 120 ohms. *Default:* `120`
loop_reversal_for_connect	Specifies how to interpret a loop reversal signal as an audio path connection indication. Set this parameter to: `DISABLED` Ignores loop reversal as an indication of audio path connection. `ENABLED` Interprets loop reversal as an indication of audio path connection. *Default:* `DISABLED`

Parameter	Value
loop_reversal_for_disconnect	Specifies how to interpret a loop reversal signal as a call disconnect indication. Set this parameter to: `DISABLED` Ignores loop reversal as a call disconnect signal. `ENABLED` Interprets loop reversal as a call disconnect signal. *Default:* `DISABLED`
max_did_digits	Specifies a value that defines the maximum number of DID digits to expect before accepting an incoming call. Set this parameter to: `0` Disables waiting for DID digits. `1 – 255` Specifies the number of digits to expect before accepting an incoming call. *Range:* `0` through `255` *Default:* `0` **Note:** The system can report all of the DID digits it received from the network to the application even if the number of received DID digits exceeds the number specified for *max_did_digits*. To remove the excess digits, set the *did_offset* parameter (see *page 1217*) so that the system only passes the expected number of digits to the application.
num_rings	Specifies a value that defines the number of rings the system must detect before the system reports a new incoming call to the application. Set this parameter to: *Range:* `1` to `255` *Default:* `2` **Note:** In North America and some other locations, the system sends the caller ID signal between the first and second rings. To detect the caller ID correctly, you must set the *num_rings* parameter to a value of 2 or greater to prevent the system from reporting the call to the application before the caller ID has been sent by the Central Office.

protocol_file	Specifies the full path and name of the protocol file to load for the E1 CAS port. Most of the time a path should be used for this file name. Set this parameter to:
	`fxo_groundstart.lec`
	`fxo_loopstart.lec`
	`fxs_groundstart.lec`
	`fxs_loopstart.lec`
	Default: `fxs_loopstart.lec`

Parameter	Value
reject_incomplete_did	Specifies the action to take when the number of DID digits received from the incoming call is less than the number of digits specified for the *max_did_digits* parameter. Set this parameter to: `FALSE` Sends an alerting or proceeding message to the network and reports the call to the application even if the number of received DID digits is less than the *max_did_digits* value. The system takes this action when the number of digits collected remains incomplete after the *did_timeout* period or when it receives a sending complete informational element (IE). `TRUE` Sends the network a reject message that causes the network to drop the call. The application does not receive any notification of the call. *Default:* `FALSE`
require_answer_signal	Specifies whether line signaling must be used to detect call answer. Set this parameter to: `FALSE` Specifies that either line signaling or call progress can detect call answer. `TRUE` Specifies that only line signaling can detect call answer (call progress only detects failed calls — for example, reorder busy). *Default:* `FALSE`
transfer_variant	Specifies the transfer method that the network (refer to the vendor specifications for your switch) runs for call transfers or disables call transfer. The SR140 does not support call transfer. Set this parameter to: `NONE` Disables call transfer. `HOOKFLASH` Specifies a hook flash transfer. *Default:* `HOOKFLASH`

Configuration Parameters for E1 CAS R2 Ports

Set one or more of the following parameters to configure an E1 port using an R2 variant of a channel associated signaling (CAS) protocol (see *page 1302* for an example configuration file).

Parameter	Value
crc	Specifies a value that indicates whether the port has cyclical redundancy checking (CRC) enabled. Set this parameter to: `DISABLED` Turns off CRC for the port. `ENABLED` Turns on CRC for the port. *Default:* `ENABLED`
default_caller_id	Specifies a string of up to 15 characters that provides the caller ID to use when placing outbound calls. If the user application provides a caller ID when placing a call, the system ignores this parameter. *Unit:* character string *Range:* `1 – 15` (`_ECC_MAX_ANI_LENGTH`) *Default:* Blank
line_coding	Specifies a value defining the type of line encoding to use for the port. Set this parameter to: `AMI` Selects Alternate Mark Inversion (AMI). `HDB3` Selects High Density Bipolar Order 3 (HDB3). *Default:* `HDB3`
line_impedance	Specifies a value that defines the line impedance the port uses. Set this parameter to: `75` Specifies that the port uses an impedance value of 75 ohms. `120` Specifies that the port uses an impedance value of 120 ohms. *Default:* `120`
max_did_digits	Specifies a value that defines the maximum number of DID digits to expect before accepting an incoming call. Set this parameter to: `0` Disables waiting for DID digits. `1 – 255` Specifies the number of digits to expect before accepting an incoming call. *Range:* `0` through `255` *Default:* `0` **Note:** The system can report all of the DID digits it received from the network to the application even if the number of received DID digits exceeds the number specified for *max_did_digits*.

Parameter	Value
protocol_file	Specifies the full path and name of the file containing the configuration for the R2 variant of a channel associated signaling (CAS) protocol. This parameter dictates which R2 CAS protocol runs on the port. Set this parameter to:

`itu_argentina.r2`	Selects the protocol file for Argentina.
`itu_brazil.r2`	Selects the protocol file for Brazil.
`itu_china.r2`	Selects the protocol file for China.
`itu_egypt.r2`	Selects the protocol file for Egypt.
`itu_korea.r2`	Selects the protocol file for Korea.
`itu_mexico.r2`	Selects the protocol file for Mexico.

Unit: character string

Range: `1 – 256` (`_MAX_PATH`)

Default: `itu_china.r2`

Configuration Parameters for E1 and T1 QSIG Ports

Set one or more of the following parameters to configure an E1 or T1 QSIG port (see *page 1304* for an example of the configuration file for a T1 QSIG port).

Parameter	Value
crc	Specifies a value that indicates whether the E1 port has cyclical redundancy checking (CRC) enabled. Set this parameter to: `DISABLED` Turns off CRC for the E1 port. `ENABLED` Turns on CRC for the E1 port. *Default:* `ENABLED`
default_caller_id	Specifies a string of up to 15 characters that provides the caller ID to use when placing outbound calls. If the user application provides a caller ID when placing a call, the system ignores this parameter. *Unit:* character string *Range:* `1 – 15` (`_ECC_MAX_ANI_LENGTH`) *Default:* Blank
did_offset	Specifies a value that defines the number of digits to remove from the beginning of the string of DID digits (see *max_did_digits on page 1224*) received from the network. If max_did_digits is set to 0, or if the number of DID digits received is less than the number specified by max_did_digits, this parameter has no effect. Set this parameter to: `0` Does not remove any DID digits. `1 – 255` Specifies the number of digits to remove. *Range:* `0` through `255` *Default:* `0`
emulation	Specifies whether the trunk acts as the primary or secondary trunk in layer 2 communication. Set this parameter to: `MASTER` Operates as the primary trunk. `SLAVE` Operates as the secondary trunk. *Default:* `SLAVE`

Parameter	Value
max_did_digits	Specifies a value that defines the maximum number of DID digits to expect before accepting an incoming call. Set this parameter to: `0` Turns off waiting for DID digits. `1 – 255` Specifies the number of digits to expect before accepting an incoming call. *Range:* `0` through `255` *Default:* `0` **Note:** The system can report all of the DID digits it received from the network to the application even if the number of received DID digits exceeds the number specified for *max_did_digits*. To remove the excess digits, set the *did_offset* parameter (see *page 1223*) so that the system only passes the expected number of digits to the application.

Advanced Configuration Parameters for an E1 or T1 QSIG Port

Change values for the following parameters only when instructed to do so by Dialogic Technical Services and Support personnel. Using different values for this protocol might produce unpredictable results.

call_diversion_completion_timer	If call diversion is turned on, the *call_diversion_completion_timer* parameter specifies the maximum duration in milliseconds that the port waits for a response from the diverted-to side of the call. If the port does not receive a response from the diverted-to side before this timer reaches the value set for it, *res.line_status* indicates a DIVERT_TO_TIME error. Set this parameter to: `0` Does not wait for a response from the diverted-to side. `1 – 20000` Specifies the maximum number of milliseconds the port waits for a response from the diverted-to side of the call. *Unit:* millisecond *Range:* `0` through `20000` *Default:* `3000` (3 seconds)

Parameter	Value
call_type	Specifies the call type to use when making the outbound call. Set this parameter to: `AUTO` Makes an outbound call using the modem type and then automatically retries the call using the voice type if the other end cannot accept modem calls. `MODEM` Makes an outbound modem (3.1 kHz audio) call. This setting provides higher quality audio for the call. However, not all numbers have this facility available. If the called party cannot accept a modem call, set the parameter to `SPEECH`. `SPEECH` Makes an outbound voice call. *Default:* `AUTO`
collision_priority	Specifies whether the local or remote end has priority in a call collision. In symmetric arrangements, call collisions can occur when both sides simultaneously transfer a SETUP message indicating the same channel or one or more channels when the transfer involves multiple channels. The local and remote ends are designated as Side A or Side B during the provisioning of your network. The "A" value has priority over the "B" value. Set this parameter to: `A` Configures port with Side A priority. `B` Configures port with Side B priority. *Default:* `B`
did_timeout	Specifies a value that defines the maximum timeout allowed before processing the call after assuming receipt of the last DID digit. Set this parameter to: `0` Indicates no waiting time. `1 – 20` Specifies the number of seconds to allow after receiving the last DID digit before processing the call. *Unit:* second *Range:* `0` through `20` *Default:* `10` (used when the Bfv API does not find another value for this parameter).
disable_alerting	Specifies whether the QSIG stack sends an alerting message on receipt of an incoming call. Setting this parameter to `TRUE` turns call alerting off and prevents the application from rejecting an incoming diverted call. Set this parameter to: `FALSE` Turns on call alerting. `TRUE` Turns off call alerting. *Default:* `FALSE`

Parameter	Value
disable_call_proceed	Specifies a value that determines whether the system sends a CALL PROCEEDING indication after receiving a SETUP message from the network. Set this parameter to: `FALSE` Indicates that the system sends a CALL PROCEEDING message after receiving a SETUP message from the network. `TRUE` Indicates that the system does not send a CALL PROCEEDING message after receiving a SETUP message from the network. *Default:* `FALSE`
disable_conn_ack	Specifies whether the system sends a connection acknowledgment message after receiving a connect message from the network. Set this parameter to: `FALSE` Allows the system to send a connection acknowledgment message after receiving a connect message from the network. `TRUE` Prevents the system from sending a connection acknowledgment message after receiving a connect message from the network. *Default:* `FALSE`
enable_call_diversion	Specifies whether the port has call diversion procedures set, allowing the port to place a call that the far end can divert to a different destination. Set this parameter to: `FALSE` Indicates that the port has call diversion procedures turned off to prevent the far end from diverting the call placed by the application. `TRUE` Indicates that the port has call diversion procedures turned on to allow the far end to divert the call placed by the application. *Default:* `TRUE`

Parameter	Value
line_build_out	Specifies one of the following values that defines the length of the telephony cable connection between the board and the T1 service (only applies to T1 ports): `0_133` Specifies a length of 0 to 133 feet. `133_266` Specifies a length of 133 to 266 feet. `266_399` Specifies a length of 266 to 399 feet. `399_533` Specifies a length of 399 to 533 feet. `533_655` Specifies a length of 533 to 655 feet. `7_5_DB` Specifies a length of negative 7.5 dB. `15_DB` Specifies a length of negative 15.0 dB. `22_5_DB` Specifies a length of negative 22.5 dB. *Default:* `0_133`
line_coding	Specifies a value defining the type of line encoding to use for the port. Set this parameter for E1 or T1 ports to: `AMI` Selects Alternate Mark Inversion (E1 or T1 digital port). `B8ZS` Selects Bipolar 8-Zero Suppression (T1 only). `HDB3` Selects High Density Bipolar Order 3 (E1 only). *Default:* `HDB3` for E1 digital ports `B8ZS` for T1 digital ports
line_impedance	Specifies a value that defines the line impedance that the port uses. Set this parameter to: `75` Specifies that the port uses an impedance value of 75 ohms. `120` Specifies that the port uses an impedance value of 120 ohms. *Default:* `120`
max_overlapped_digits	Specifies the maximum number of digits to send when the application supports overlapped dialing for long phone numbers. Set this parameter to: `0` Disables support for overlapped dialing. *Range:* `1 – 24 (IISDN_MAX_DIGITS(24))` *Default:* `20`
numbering_plan	Specifies a value that identifies the type of numbering plan used for outbound calls (called party number). Set this parameter to: `ISDN` Indicates that the port uses an ISDN numbering plan. `PRIVATE` Indicates that the port uses a private numbering plan. `UNKNOWN` Indicates that the port uses an unknown numbering plan. *Default:* `UNKNOWN`

Parameter	Value

numbering_type	Specifies a value that identifies the type of telephone number used for outbound calls (called party number). Set this parameter to:
	`INTERNATIONAL`
	Indicates that the port uses an international numbering type (valid if *numbering_plan* is set to `ISDN`).
	`NATIONAL`
	Indicates that the port uses a national (North American) numbering type (valid if *numbering_plan* is set to `ISDN`).
	`SUBSCRIBER`
	Indicates that the port uses a subscriber numbering type (valid if *numbering_plan* is set to `ISDN`).
	`LEVEL_2_REGION`
	Indicates that the port uses a level 2 regional numbering type (valid if *numbering_plan* is set to `PRIVATE`).
	`LEVEL_1_REGION`
	Indicates that the port uses a level 1 regional numbering type (valid if *numbering_plan* is set to `PRIVATE`).
	`LEVEL_0_REGION`
	Indicates that the port uses a level 0 regional numbering type (valid if *numbering_plan* is set to `PRIVATE`).
	`PISN_SPECIFIC`
	Indicates that the port uses a Private Integrated Services Network (PISN) numbering type (valid if *numbering_plan* is set to `PRIVATE`).
	`UNKNOWN`
	Indicates that the port uses an unknown numbering type.
	Default: `UNKNOWN`

Parameter	Value

presentation	Specifies a value that indicates the type of presentation of the calling party number the port uses when placing an outbound call. Set this parameter to:
	`ALLOWED`
	Indicates that the port allows presentation of the calling party number to the called party.
	`APP_DEFINED`
	Indicates that the application passes in the value for this parameter.
	`NUM_NOT_AVAIL`
	Indicates that the port does not have a calling party number specified to present to the called party.
	`RESTRICTED`
	Indicates that the port restricts presentation of the calling party number to specific called party numbers.
	Note: You should set the parameter to `NUM_NOT_AVAIL` when connected to a public network.
	Default: `ALLOWED`

qsig_support	Specifies the standard to use for coding QSIG supplementary services messages.
	The ECMA standard for coding the different supplementary services has evolved, changing some services (for example, call diversion, call transfer, and name identification) from encoding the ASN.1 operational value as an OBJECT IDENTIFIER to an INTEGER. Some PBXs (Siemens) continue to use the old standard while others use the new standard. Set this parameter to specify whether the system codes the supplementary services messages according to the old or new standard.
	`OLD`
	Encodes operational value as an OBJECT IDENTIFIER according to the old version of the ECMA standard.
	`NEW`
	Encodes operational value as an INTEGER according to the new version of the ECMA standard.
	Default: `NEW`

Parameter	Value
reject_incomplete_did	Specifies the action to take when the number of DID digits received from the incoming call is less than the number of digits specified for the *max_did_digits* parameter. Set this parameter to: `FALSE` Sends an alerting or proceeding message to the network and reports the call to the application even if the number of received DID digits is less than the *max_did_digits* value. The system takes this action when the number of digits collected remains incomplete after the *did_timeout* period or when it receives a sending complete informational element (IE). `TRUE` Sends the network a reject message that causes the network to drop the call. The application does not receive any notification of the call. *Default:* `FALSE`
request_aoc	Specifies a value that indicates whether the port follows the protocol's Advice of Charge (AOC) procedures. Set this parameter to: `FALSE` Indicates that the port does not follow the protocol's AOC procedures. `TRUE` Indicates that the port uses the protocol's AOC procedures. *Default:* `FALSE`
sabme	Specifies whether the port sends layer 2 Set Asynchronous Balanced Mode Extended (SABME) messages. Set this parameter to: `FALSE` Indicates that the port does not send SABME messages. `TRUE` Indicates that the port sends layer 2 SABME messages. *Default:* `TRUE`

Parameter	Value

screening	Specifies a value that indicates whether the port provides and validates the calling party number passed to the called party. Set this parameter to:
	`APP_DEFINED`
	Indicates that the application passes in the value for this parameter.
	`NETWORK_PROVIDED`
	Indicates that the network validates the calling party number.
	`USER_NOT_SCREENED`
	Indicates that the port provides the calling party number without validating it.
	`USER_VERIFICATION_PASSED`
	Indicates that the port provides the calling party number and passes a successfully validated number to the called party.
	Note: You should set the parameter to `USER_NOT_SCREENED` when connected to a public network.
	Default: `USER_NOT_SCREENED`

send_dialcomplete	Specifies whether the system sends an informational element (IE) for outbound calls that indicates the end of dialing. Set this parameter to: `FALSE` Indicates that the port does not transmit a DIAL COMPLETE message on outbound calls. `TRUE` Requests the system to transmit a DIAL COMPLETE message on outbound calls. *Default:* `TRUE`
transfer_variant	Specifies the transfer method that the network runs for call transfers or disables call transfer. The SR140 does not support call transfer. Set this parameter to: `NONE` Disables call transfer. `QSIG` Specifies QSIG transfer protocol. *Default:* `QSIG`

Configuration Parameters for T1 ISDN Ports

Set one or more of the following parameters to configure a T1 ISDN port (see *page 1303* for an example configuration file).

Parameter	Value
ani_prefix	Specifies a value for the prefix before the ANI digits. *Unit:* character string *Range:* 1 – 15 (_ECC_MAX_ANI_LENGTH) *Default:* None
call_type	Specifies the call type to use when making the outbound call. Use one of the following values for this parameter: AUTO — Makes a call using the modem type and then automatically retries the call using the voice type if the other end cannot accept modem calls. MODEM — Makes a modem (3.1 kHz audio) call. This setting provides higher quality audio for the call. SPEECH — Makes a voice call. *Default:* AUTO
default_caller_id	Specifies a string of up to 15 characters that provides the caller ID to use when placing outbound calls. If the user application provides a caller ID when placing a call, the system ignores this parameter. *Unit:* character string *Range:* 1 – 15 (_ECC_MAX_ANI_LENGTH) *Default:* Blank
did_offset	Specifies a value that defines the number of digits to remove from the beginning of the string of DID digits (see *max_did_digits on page 1234*) received from the network. If max_did_digits is set to 0, or if the number of DID digits received is less than the number specified by max_did_digits, this parameter has no effect. Set this parameter to: 0 — Does not remove any DID digits. 1 – 255 — Specifies the number of digits to remove. *Range:* 0 through 255 *Default:* 0

Parameter	Value
disable_conn_ack	Specifies whether the system sends a connection acknowledgment after receiving a connect message from the network. Set this parameter to: `FALSE` The system sends a connection acknowledgment message after receiving a connect message from the network. `TRUE` The system does not send a connection acknowledgment message after receiving a connect message from the network. *Default:* `FALSE`
dnis_prefix	Specifies a value for the prefix before the DNIS digits. *Unit:* character string *Range:* `1 – 15` (`_ECC_MAX_ANI_LENGTH`) *Default:* None
emulation	Specifies whether to configure the trunk for Central Office (CO) or Customer Premise Equipment (CPE) protocol emulation. Use this parameter for testing purposes only. `CO` Emulates the CO protocol. `CPE` Emulates the CPE protocol. *Default:* `CPE`
line_build_out	Specifies one of the following values that defines the length of the telephony cable connection between the board and the T1 service: `0_133` Specifies a length of 0 to 133 feet. `133_266` Specifies a length of 133 to 266 feet. `266_399` Specifies a length of 266 to 399 feet. `399_533` Specifies a length of 399 to 533 feet. `533_655` Specifies a length of 533 to 655 feet. `7_5_DB` Specifies a length of negative 7.5 dB. `15_DB` Specifies a length of negative 15.0 dB. `22_5_DB` Specifies a length of negative 22.5 dB. *Default:* `0_133`
line_coding	Specifies a value defining the type of line encoding to use for the port. Set this parameter to: `AMI` Selects Alternate Mark Inversion (AMI). `B8ZS` Selects Bipolar 8-Zero Suppression (B8ZS). `JBZS` Selects Jammed Bit Zero Suppression. `ZBTSI` Selects Zero Byte Time Slot Interchange. *Default:* `B8ZS`

Parameter	Value
max_did_digits	Specifies a value that defines the maximum number of DID digits to expect before accepting an incoming call. Set this parameter to: `0` Disables waiting for DID digits. `1 – 255` Specifies the number of digits to expect before accepting an incoming call. *Range:* `0` through `255` *Default:* `0` **Note:** The system can report all of the DID digits it received from the network to the application even if the number of received DID digits exceeds the number specified for *max_did_digits*. To remove the excess digits, set the *did_offset* parameter (see *page 1232*) so that the system only passes the expected number of digits to the application.
NSF	Specifies a value indicating that the user's call setup message, if defined, includes a network specific facility (NSF) message. Set the value in this parameter to one of the following to indicate the type of service used to send the NSF message in the outbound call setup: `0` Indicates that the call setup does not include an NSF message. `1` Indicates use of an AT&T software-defined network or a Northern Telecom private network. `2` Indicates use of the AT&T Megacom 800 service. `3` Indicates use of the AT&T Megacom or Northern Telecom OutWATS service. `4` Indicates use of the Northern Telecom foreign exchange service. `5` Indicates use of the Northern Telecom tie trunk service. `6` Indicates use of the AT&T Accunet service. `8` Indicates use of the AT&T international 800 service. `16` Indicates use of the Northern Telecom Trunk Optimization (TRO) call service. *Default:* `0`

Parameter	Value

numbering_plan	Specifies a value that identifies the type of numbering plan used for outbound calls (called party number). Set this parameter to:
	`ISDN`
	Indicates that the port uses an ISDN numbering plan.
	`PRIVATE`
	Indicates that the port uses a private numbering plan.
	`TELEPHONY`
	Indicates that the port uses a telephony numbering plan.
	`UNKNOWN`
	Indicates that the port uses an unknown numbering plan.
	Default: `UNKNOWN`

numbering_type	Specifies a value that identifies the type of telephone number used for outbound calls (called party number). Set this parameter to:
	`ABBREVIATED`
	Indicates that the port uses an abbreviated numbering type.
	`INTERNATIONAL`
	Indicates that the port uses an international numbering type.
	`NATIONAL`
	Indicates that the port uses a national (North American) numbering type.
	`SUBSCRIBER`
	Indicates that the port uses a subscriber numbering type.
	`UNKNOWN`
	Indicates that the port uses an unknown numbering type.
	Default: `UNKNOWN`

Parameter	Value

presentation	Specifies a value that indicates the type of presentation of the calling party number the port uses when placing an outbound call. Set this parameter to:
	`ALLOWED`
	Indicates that the port allows presentation of the calling party number to the called party.
	`NUM_NOT_AVAIL`
	Indicates that the port does not have a calling party number specified to present to the called party.
	`RESTRICTED`
	Indicates that the port restricts presentation of the calling party number to specific called party numbers.
	Note: You should set the parameter to `NUM_NOT_AVAIL` when connected to a public network.
	Default: `ALLOWED`

protocol	Specifies the type of protocol variant to use for the port. Set this parameter to:
	`ATT`
	AT&T as defined in AT&T PUB 41449.
	`ISDN1`
	Bellcore National (North American) ISDN-1 Standard.
	`ISDN2`
	Bellcore National (North American) ISDN-2 Standard (TR-NWT-001268).
	`Jate`
	Jate (Japan) INS-1500 standard.
	`Nortel`
	Northern Telecom as defined in NIS A211-1.
	`CTR4`
	NET-5 standard for PRI connections throughout Europe (also referred to as Euro-ISDN). Choosing this variant changes the Layer 2 protocol timers to their appropriate NET-5 defaults.
	`CCITT`
	General ITU-T Q.931 conformance.
	Default: `ATT`

Parameter	Value
reject_incomplete_did	Specifies the action to take when the number of DID digits received from the incoming call is less than the number of digits specified for the *max_did_digits* parameter. Set this parameter to: `FALSE` Sends an alerting or proceeding message to the network and reports the call to the application even if the number of received DID digits is less than the *max_did_digits* value. The system takes this action when the number of digits collected remains incomplete after the *did_timeout* period or when it receives a sending complete informational element (IE). `TRUE` Sends the network a reject message that causes the network to drop the call. The application does not receive any notification of the call. *Default:* `FALSE`
sabme	Specifies whether the port sends layer 2 Set Asynchronous Balanced Mode Extended (SABME) messages. Set this parameter to: `FALSE` Indicates that the port does not send SABME messages. `TRUE` Indicates that the port sends layer 2 SABME messages. *Default:* `TRUE` (default when `emulation = CPE`) `FALSE` (default when `emulation = CO`)

screening	Specifies a value that indicates whether the port provides and validates the calling party number passed to the called party. Set this parameter to:
	`NETWORK_PROVIDED`
	Indicates that the network validates the calling party number.
	`USER_NOT_SCREENED`
	Indicates that the port provides the calling party number without validating it.
	`USER_VERIFICATION_FAILED`
	Indicates that the port failed to validate the calling party number.
	`USER_VERIFICATION_PASSED`
	Indicates that the port provides the calling party number and passes a successfully validated number to the called party.
	Note: You should set the parameter to `USER_NOT_SCREENED` when connected to a public network.
	Default: `USER_NOT_SCREENED`

Parameter	Value

switch_type	Specifies a value indicating the type of switch used for the board connection. Set this parameter to:
	`ATT_4ESS`
	AT&T #4 ESS.
	`ATT_5ESS`
	AT&T #5 ESS.
	`NTI_DMS100`
	Nortel DMS-100.
	`NTI_DMS250`
	Nortel DMS-250.
	`MD110_T1`
	Selects Ericsson MD-110 switch for North America.
	`MD110_E1`
	Selects Ericsson MD-110 switch (International).
	`SIEMENS`
	Siemens.
	`NTT`
	Japan.
	`UNKNOWN`
	Selects a switch type that complies with the ITU-T standards.
	Default: `ATT_4ESS`

Parameter	Value
transfer_variant	Specifies the transfer method that the network (refer to the vendor specifications for your switch) runs for call transfers or disables call transfer. Set this parameter to: The SR140 does not support call transfer. `NONE` Disables call transfer. `TBCT` Specifies a Bellcore National (North American) ISDN Two B-Channel Transfer (TBCT) method. `RLT` Specifies a Release Link Trunk transfer method. `NTT` Specifies a JATE active redirecting transfer method. `NTT_MP` Specifies a JATE active redirecting transfer for a point to multipoint configuration. *Default:* `TBCT` **Note:** Set the value to `NTT` when you set `country_code = JAPAN` in the *btcall.cfg* user-defined configuration file (see *page 1155*). Set the value to `RLT` when you set the *switch_type* value to `NTI_DMS100` or `NTI_DMS250` (see *page 1238*).
wait_for_bchannel_status	Specifies a value that determines when the system puts the B-channels in service. Set this parameter to: `FALSE` Requests that the system puts all the B-channels in service as soon as the D-channel is in service. `TRUE` Requests that the system waits for the network to specifically enable each B-channel on the trunk before allowing the application to use a B-channel. *Default:* Depends on the value set in the *switch_type* parameter.
wait_for_conn_ack	Specifies whether the system waits for the network to acknowledge a connect request before notifying the application that a call has been answered. Set this parameter to: `FALSE` Specifies that the system does not wait for the network to acknowledge a connect request before advancing an incoming call to the connected state. `TRUE` Requests the system to wait for the network to acknowledge a connect request before advancing an incoming call to the connected state. *Default:* `FALSE`

Configuration Parameters for T1 RBS Ports

Set one or more of the following parameters to configure a port that uses a T1 robbed bit signaling (RBS) protocol (see *page 1306* for an example configuration file).

Parameter	Value
caller_id	Specifies whether detection of V.23-based caller ID has been turned on. Set this parameter to: `FALSE` Turns off detection of V.23-based caller ID for the port. `TRUE` Turns on detection of V.23-based caller ID for the port. **Note:** In North America and some other locations, the system sends the caller ID signal between the first and second rings. To detect the caller ID correctly, you must set the *num_rings* (*page 1199*) parameter to a value of 2 or greater to prevent the system from reporting the call to the application before the caller ID has been sent by the Central Office. *Default:* `TRUE`
did_offset	Specifies a value that defines the number of digits to remove from the beginning of the string of DID digits (see *max_did_digits on page 1242*) received from the network. If max_did_digits is set to 0, or if the number of DID digits received is less than the number specified by max_did_digits, this parameter has no effect. Set this parameter to: `0` Does not remove any DID digits. `1 – 255` Specifies the number of digits to remove. *Range:* `0` through `255` *Default:* `0`
did_timeout	Specifies a value that defines the maximum timeout allowed before processing the call after assuming receipt of the last DID digit. Set this parameter to: `0` Indicates no waiting time. `1 – 20` Specifies the number of seconds to allow after receiving the last DID digit before processing the call. *Unit:* second *Range:* `0` through `20` *Default:* `10` (used when the Bfv API does not find another value for this parameter).

Parameter	Value
flash_hook_duration	Specifies a value for the duration of a flash hook signal. This parameter defines the amount of time to place the line on hook (loop current dropped) during a flash hook. Set the value in units of 10 ms. *Unit:* 10 ms *Range:* `1 – 500` *Default:* `50`
line_build_out	Specifies one of the following values that defines the length of the telephony cable connection between the board and the T1 service: `0_133` Specifies a length of 0 to 133 feet. `133_266` Specifies a length of 133 to 266 feet. `266_399` Specifies a length of 266 to 399 feet. `399_533` Specifies a length of 399 to 533 feet. `533_655` Specifies a length of 533 to 655 feet. `7_5_DB` Specifies a length of negative 7.5 dB. `15_DB` Specifies a length of negative 15.0 dB. `22_5_DB` Specifies a length of negative 22.5 dB. *Default:* `0_133`
line_coding	Specifies a value defining the type of line encoding to use for the port. Set this parameter to: `AMI` Selects Alternate Mark Inversion (AMI). `B8ZS` Selects Bipolar 8-Zero Suppression (B8ZS). `JBZS` Selects Jammed Bit Zero Suppression. `ZBTSI` Selects Zero Byte Time Slot Interchange. *Default:* `B8ZS`
line_type	Specifies a value defining the type of framing to use for the port. Set this parameter to: `D4` Selects AT&T D4 framing format. `ESF` Selects Extended Super Frame (ESF). *Default:* `ESF`
loop_reversal_for_connect	Specifies how to interpret a loop reversal signal as an audio path connection indication. Set this parameter to: `DISABLED` Ignores loop reversal as an indication of audio path connection. `ENABLED` Interprets loop reversal as an indication of audio path connection. *Default:* `DISABLED`

Parameter	Value
loop_reversal_for_disconnect	Specifies how to interpret a loop reversal signal as a call disconnect indication. Set this parameter to: `DISABLED` Ignores loop reversal as a call disconnect signal. `ENABLED` Interprets loop reversal as a call disconnect signal. *Default:* `DISABLED`
max_did_digits	Specifies a value that defines the maximum number of DID digits to expect before accepting an incoming call. Set this parameter to: `0` Turns off waiting for DID digits. `1 – 255` Specifies the number of digits — use a number in this range for all countries except Japan. `1 – 4` Specifies the number of digits for Japan only. *Range:* `0` through `255` for all except Japan; `0` through `4` for Japan. *Default:* `0` **Note:** The system only reports the expected number of DID digits (the value specified for *max_did_digits*) to the application even if the number of DID digits it received from the network exceeds the number specified for *max_did_digits*.
num_rings	Specifies a value that defines the number of rings the system must detect before the system reports a new incoming call to the application. *Range:* 1 to 255 *Default:* `2`

protocol_file	Specifies the name of the T1 robbed-bit signaling (RBS) protocol file to load for the port. This parameter dictates the protocol that runs on the port. Set this parameter to:
	`fxo_groundstart.lec`
	`fxo_loopstart.lec`
	`fxs_groundstart.lec`
	`fxs_loopstart.lec`
	`immediatedial.lec`
	`winkstart.lec`
	Default: `winkstart.lec`

Parameter	Value
reject_incomplete_did	Specifies the action to take when the number of DID digits received from the incoming call is less than the number of digits specified for the *max_did_digits* parameter. Set this parameter to: `FALSE` Reports the call to the application even if the number of received DID digits is less than the *max_did_digits* value. The system takes this action when the number of digits collected remains incomplete after the *did_timeout* period. `TRUE` Sends the network a reject message that causes the network to drop the call. The application does not receive any notification of the call. *Default:* `FALSE`
require_answer_signal	Specifies whether line signaling must be used to detect call answer. Set this parameter to: `FALSE` Specifies that either line signaling or call progress can detect call answer. `TRUE` Specifies that only line signaling can detect call answer (call progress only detects failed calls — for example, reorder busy). *Default:* `FALSE`
transfer_variant	Specifies the transfer method that the network (refer to the vendor specifications for your switch) runs for call transfers or disables call transfer. Set this parameter to: The SR140 does not support call transfer. `NONE` Disables call transfer. `HOOKFLASH` Specifies a hook flash transfer. *Default:* `HOOKFLASH`

Other Parameters

In addition to the configuration-specific parameters for a T1 RBS port, the Bfv API also uses parameters from the *BT_CPARM.CFG* file to configure lines. See *BT_CPARM.CFG Parameter File* on *page 1441* for descriptions and values for the following parameters.

- *dial_tone_min*
- *dtone_len*
- *dtone_timeout_highbyte*
- *dtone_timeout_lowbyte*
- *loopcur_timeout*
- *loop_seizure*
- *loop_max_break*
- *max_interdigit*
- *min_on_hook*
- *pre_wink*
- *post_wink*
- *pulse_break*
- *pulse_inter_time*
- *pulse_make*
- *pulse_max_break*
- *pulse_min_break*
- *ring_blank*
- *ring_len*
- *tone_inter_time*
- *tone_len*

Internet Protocol (IP) Call Control Configuration Parameters

The following paragraphs describe the sections of the call control configuration file that configure your modules and the Bfv API to use an internet protocol (IP) call control stack. See *page 1307* for example configuration files.

Note: Only the TR1034 and the SR140 support these configuration parameters.

These sections of the configuration file include:

host_module.#

Provides parameters to define an IP call control stack for the Bfv API to use.

This configuration section also allows you to configure:

- T.38 fax transport parameters for a module.
- Custom key-value pairs for the IP call control stack to read from the configuration file.
- RTP parameters for the Bfv API

module.#/ethernet.#

Provides parameters to define an Ethernet interface.

module.#/host_cc.#

Provides parameters to define an IP call control stack for the module to use.

Configuring An IP Call Control Stack For Bfv API

In the `host_module.#` section of the configuration file, identify each IP call control stack that the Bfv API can use. Note the following:

1. You must create a `host_module.#` section for each IP call control stack your application uses.
2. Number each IP call control stack uniquely, starting at 1.
3. You can add up to 9 host modules.

Set the following parameters for each `host_module.#` section of the call control configuration file.

Parameter	Value
enabled	Specifies whether the Bfv API can use the IP call control stack. Setting this parameter to: `FALSE` Makes the named IP call control stack unavailable to the Bfv API. `TRUE` Makes the named IP call control stack available for the Bfv API to use. *Value Type:* Boolean *Default:* `TRUE`
module_library	Specifies the full path and filename of the IP call control stack. The Bfv API attempts to load this library dynamically. Set this parameter to: `FULLPATH` Contains the full path to the named library containing the IP call control stack. *Value Type:* character string *Default:* None
media_use_ unique_port	Specifies whether the system reuses an IP port when switching from one media type to another. **Note:** You should contact Dialogic Technical Services and Support before attempting to use this media switching parameter. Set this parameter to: `FALSE` Selects the same IP port when switching to a new media type. `TRUE` Selects a different IP port when switching to a new media type. *Value Type:* Boolean *Default:* `FALSE` (reuse IP port).

Parameter	Value
routing_table	Optional parameter to specify the full path and filename of the routing table configuration file. This file contains one or more routing rules and is used by the SIP IP call control protocol stack to route inbound calls. FULLPATH Contains the full path to the routing table configuration file. *Value Type:* character string *Default:* `None`

Configuring T.38 Fax Transport Parameters

Specify values for the following T38-specific parameters in the `host_module.#/t38parameters` section of the call control configuration file.

Parameter	Value
fax_transport_protocol	Specifies the method for transporting fax media. t38_never — Fax will use G.711 pass-through only. t38_only — Fax will use T.38 only and the call will fail if T.38 cannot be negotiated. t38_first — Fax will attempt T.38 and fall back to G.711 pass-through if T.38 cannot be negotiated. *Value Type:* Character string *Default:* t38_only
media_renegotiate_delay_inbound	Controls media renegotiation to image (T.38) on inbound calls. If the gateway is responsible for media renegotiation, set this parameter to -1 to disable initiating the media renegotiation to image. If the UAC is responsible for media renegotiation to image, set this parameter to a value between 0 and 60000. Numbers greater than 0 indicate the number of milliseconds to delay before attempting media renegotiation. The time difference between inbound and outbound media renegotiate delay should be at least 500 ms. A value of 0 will cause an immediate renegotiation, while -1 will wait for a renegotiation to image. Set this parameter to: `–1` — Disables media renegotiation on inbound calls. `0` — Does not delay before attempting to renegotiate the media. `>0` — Waits this number of milliseconds before attempting to renegotiate the media. **Note:** Set at least 500 ms of time difference between inbound and outbound media renegotiate delay parameters. *Unit:* ms *Range:* `–1` and `0` to `60000` *Value Type:* decimal *Default:* `1000` (1 second)

Parameter	Value
media_renegotiate_delay_outbound	Controls media renegotiation to image (T.38) on outbound calls. If the gateway is responsible for media renegotiation, set this parameter to -1 to disable initiating the media renegotiation to image. If the UAC is responsible for media renegotiation to image, set this parameter to a value between 0 and 60000. Numbers greater than 0 indicate the number of milliseconds to delay before attempting media renegotiation. The time difference between inbound and outbound media renegotiate delay should be at least 500 ms. A value of 0 will cause an immediate renegotiation, while - 1 will wait for a renegotiation to image. Set this parameter to: `–1` Disables media renegotiation on outbound calls. `0` Does not delay before attempting to renegotiate the media. `>0` Waits this number of milliseconds before attempting to renegotiate the media. **Note:** Set at least 500 ms of time difference between inbound and outbound media renegotiate delay parameters. *Unit:* ms *Range:* `–1` and `0` to `60000` *Value Type:* decimal *Default:* `–1`
media_passthrough_timeout_outbound	Sets the timer to fail over to fax passthrough when no T.38 is negotiated on outbound calls. This timer is active only when [media_renegotiate_delay_outbound] is set to -1, [fax_transport_protocol] is set to t38_first, and the module supports fax passthrough. Numbers greater than 0, indicate the number of milliseconds to wait for T.38 negotiation before performing fax passthrough. A value of 0 will cause an immediate renegotiation to passthrough, while -1 will suppress renegotiation to fax passthrough. Set this parameter to: `–1` Suppress renegotiation to fax passthrough. `0` Cause an immediate renegotiation to passthrough. `>0` Number of milliseconds to wait for T.38 negotiation before performing fax passthrough. *Unit:* ms *Range:* `–1` and `0` to `60000` *Value Type:* decimal *Default:* 4000

Parameter	Value
media_passthrough_timeout_inbound	Sets whether media renegotiation will be attempted before doing fax passthrough on inbound calls. Set [media_renegotiate_delay_inbound] to -1 to disable initiating the media renegotiation to image and [fax_transport_protocol] to t38_first. The module also must support fax passthrough. Numbers greater than 0, indicate the number of milliseconds to delay before attempting media renegotiation if the module supports fax passthrough. Set this parameter to: `-1` Suppress renegotiation to fax passthrough. `0` Use the default number of milliseconds to wait for T.38 negotiation before performing fax passthrough. `>0` Number of milliseconds to wait for T.38 negotiation before performing fax passthrough. *Unit:* ms *Range:* `-1` and `0` to `60000` *Value Type:* decimal *Default:* 1000
t38_fax_fill_bit_removal	Specifies whether the Bfv API can remove or insert fill bits to reduce the bandwidth of the transport mechanism. Set this parameter to: `FALSE` Indicates that the Bfv API does not support the capability. `TRUE` Indicates that the Bfv API can remove or insert fill bits. *Value Type:* Boolean *Default:* `FALSE` **Note:** This parameter does not affect the normal T.30-level capability to remove or insert fill bits.
t38_fax_rate_management	Specifies a value that identifies the data rate management method of the transport. Set this parameter to: `localTCF` Indicates that the transport uses the local training check frame (TCF) data rate management type (not supported). `transferredTCF` Indicates that the transport uses the transferred training check frame (TCF) data rate management type. *Value Type:* character string *Default:* `transferredTCF`

Parameter	Value
t38_fax_transcoding_JBIG	Specifies whether the Bfv API can convert to and from JBIG fax images to reduce the bandwidth of the transport mechanism when using a reliable transport (for example, TCP). Set this parameter to: `FALSE` Indicates that the Bfv API does not support the capability. `TRUE` Indicates that the Bfv API can convert JBIG fax images. *Value Type:* Boolean *Default:* `FALSE`
t38_fax_transcoding_MMR	Specifies whether the Bfv API can convert to and from MMR fax compression to reduce the bandwidth of the transport mechanism when using a reliable transport (for example, TCP). Set this parameter to: `FALSE` Indicates that the Bfv API does not support the capability. `TRUE` Indicates that the Bfv API can convert MMR compression. *Value Type:* Boolean *Default:* `FALSE` **Note:** This parameter does not affect the normal T.30-level capability to use MMR if the two endpoints select MMR as a line compression format.
t38_fax_udp_ec	Specifies a value that identifies the error correction method of the T.38 fax transport. Set this parameter to: `t38UDPFEC` The transport uses the T.38 user datagram protocol (UDP) forward error correction (FEC) method (not supported). `t38UDPRedundancy` The transport uses the T.38 UDP redundancy error correction method. *Value Type:* character string *Default:* `t38UDPRedundancy`
t38_stream_renegotiation	Specifies how the T.38 offer will be composed. `single` The T.38 offer will be composed of only one media stream. `replace` The T.38 offer will replace the active stream. `append` The T.38 offer will be appended to the end in a new media stream. *Value Type:* character string *Default:* `single`

Parameter	Value

rtp_ced_enable	Specifies whether to play the CED/ANSam tone for inbound IP calls. If set to true, channels will generate CED/ANSam tone using the RTP protocol for SIP and H.323 fax calls which do not immediately start as a T.38 fax call. If set to false, the CED/ANSam tone is not generated.
	`FALSE` `CED/ANSam tone is not generated`
	`TRUE` `Channels generate CED/ANSam tone`
	Value Type: Boolean
	Default: `TRUE`
	
	Note: Setting this parameter to true can cause some gateways to attempt an RTP fax rather than a T.38 fax.

t38_fax_version	Controls the maximum T.38 ASN.1 version the IP Call Control offers or accepts from a remote party. Versions 0, 1, 2 support a maximum bit rate of 14,400 bps. Version 3 supports V.34 and the following are the possible bit rates: 33,600 (default), 31,200, 28,800, 26,400, 24,000, 21,600, 16,800 Must be version 1 or higher in order to support T.38 Internet Aware Fax (IAF) modulation. *Unit:* not applicable *Range:* 0,1,2,3 *Value Type:* decimal *Default:* 0
t38_t30_fastnotify	Specifies whether the transport signals the beginning of T.30 by means of a zero-length data field or uses a T.30 indicator value. Set this parameter to: `FALSE` Indicates that the T.38 fax transport uses a zero-length data field to signal the beginning of T.30. `TRUE` Indicates that the transport uses a `T30_INDICATOR` value to signal the beginning of T.30. *Value Type:* Boolean *Default:* `FALSE`

Parameter	Value
t38_UDPTL_redundancy_depth_control	Specifies a value that defines the number of prior messages to include as redundancy messages in a transmitted UDPTL packet carrying signal information (FSK signals). Set this parameter to: 0 – 5 Specifies a number value defining how many prior messages to include as redundancy messages in a packet carrying control data. *Unit:* number *Range:* 0 through 5 *Value Type:* decimal *Default:* 5
t38_UDPTL_redundancy_depth_image	Specifies a value that defines the number of prior messages to include as redundancy messages in a transmitted UDPTL packet carrying image data. Set this parameter to: 0 – 2 Specifies a number value defining how many prior messages to include as redundancy messages in a packet carrying image data. *Unit:* number *Range:* 0 through 2 *Value Type:* decimal *Default:* 2
t38_type_of_service	Determines how the first six bits of the ToS DCSP (Differentiated Services Point Code) field in the IP header are set for T.38 packets. This parameter is available for Linux only. For Windows, the value of the DSCP bits is set from a group policy. *Unit*: none *Range*: 0 - 63 *Value Type*: decimal *Default*: 0

Parameter

t38_max_bit_rate

Value

If a remote T.38 terminal specifies a maximum bit rate that differs from the setting specified by this parameter, the actual maximum bit rate selected for a call will be negotiated to the lower of the two values with the following exceptions:

Remote T.38 Maximum Bit Rate	Negotiated T.38 Maximum Bit Rate
0	Value specified by this parameter
<2400	2400
>33600	Value specified by this parameter
2400 < Bit Rate <33600 but not one of the values listed in the Range section below	If a T.38 Maximum Bit Rate value specified by a remote T.38 terminal is between 2400 and 33600 but is not one of the values listed in the Range section below, the negotiated maximum bit rate value will be the next lowest supported value. For example, if a remote T.38 terminal specifies a maximum bit rate value of 20000, the negotiated T.38 maximum bit rate value will be the lower of either 19200 or the value specified by this parameter.

Unit: bits per second

Range: 2400, 4800, 7200, 9600, 12000, 14400, 16800 19200, 21600, 24000, 26400, 28800, 31200, 33600

Value Type: decimal

Default: 14400

Parameter	Value
t38_fax_max_buffer	Specifies value for T38FaxMaxBuffer attribute of T.38 codec. This parameter is used to specify the maximum number of octets that can be stored before an overflow condition occurs. *Unit*: octets *Range*: 64 through 65535 *Value Type*: decimal *Default*: 200
t38_fax_max_datagram_recv	Specifies size of maximum datagram that can be transmitted for T.38. This value or the maximum datagram size exchanged in the SDPs via T38FaxMaxDatagram will be used to set the maximum datagram size that can be transmitted for T.38, whichever value is smaller. See the t38_fax_max_datagram_send parameter. *Unit*: octets *Range*: 1 through 65535 *Value Type*: decimal *Default*: 125
t38_fax_max_datagram_send	Specifies value for T38FaxMaxDatagram attribute for T.38 in the SDP. This value or the maximum datagram size from the remote side will be used for the T38FaxMaxDatagram attribute for T.38 which is sent in the SDP, whichever value is smaller. *Unit*: octets *Range*: 1 through 65535 *Value Type*: decimal *Default*: 72

Parameter	Value
t38_fax_max_datagram_iaf	Specifies size of maximum datagram packets that can be transmitted or received for T.38 Internet Aware Fax (IAF) transfers. This parameter is used to specify the maximum size of a datagram packet that can be transmitted or received for T.38 in IAF mode. *Unit*: octets *Range*: 1 through 65535 *Value Type*: decimal *Default*: 270
g711_fallback_rtp_reinvite	Specifies whether or not a SIP RTP reINVITE should be transmitted for G.711 fallback mode if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable). Set this parameter to: `FALSE` Do not transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected. `TRUE` Transmit a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected. *Value Type*: Boolean *Default*:FALSE Setting this field to a value of *TRUE* will result in transmission of a SIP RTP reINVITE if a SIP T.38 reINVITE is rejected with either a 488 (Not Acceptable Here) or a 606 (Not Acceptable) and the fax transport protocol (*fax_transport_protocol*) parameter is set to *t38_first*. The SDP settings in the SIP RTP reINVITE will be the same RTP codec settings initially used to establish the call. **Note:** This parameter only works for calls using the SIP internet protocol and will be ignored for all calls using the H.323 internet protocol or PSTN line types.

Configuring Customer Parameters for a Third Party IP Call Control Stack

The Bfv API also provides a mechanism for third party IP call control stacks to read custom key-value pairs from the call control configuration file. The keys must be unique within the particular stack. The Bfv API supports both numerical and character string values.

Specify the value type when the third party IP call control stack places a request to query the parameters in the call control configuration file.

The Bfv API does not provide any validation of the key-value pairs, although the stack can return an error during initialization if it detects one or more invalid parameters.

Add the key-value pairs to the `host_module.#/parameters` section of the call control configuration file as shown in the following example:

```
[host_module.1]
    module_library=c:\vendor\vendors_sip_stack.dll
    enabled=true
[host_module.1/parameters]
    ModuleString1=value
    ModuleNumber1=23
    ModuleString2=value
    ModuleNumber2=24
    ModuleString3=value
    ModuleNumber3=25
```

The third party IP call control stack can read stack specific keys at initialization time.

To make it easier to configure third party IP call control stacks, the Bfv API has predefined a set of common parameters for several IP protocols. Stack authors must review this list of parameters to see if a needed parameter already exists before creating a new one. See:

- *Table 24 on page 1258 for H.323 parameters.*
- *Table 26 on page 1267 for Basic SIP parameters*
- *Table 27 on page 1271 for Advanced SIP parameters*

The maximum length of a key's name is defined as `CIPI_MAX_KEY_NAME`, and the maximum length of a character string value is defined as `CIPI_MAX_KEY_VALUE`. These lengths must allow for the `NULL` terminating character.

Table 24. Basic Predefined H.323 IP Call Control Stack Parameters

Key Name	Description
h323_default_gateway	Indicates the IP address of a default gateway to use for outbound calls. If a user only specifies a phone number when making an H.323 call and the application is not using an H.323 gatekeeper, the Bfv API forwards the call to the gateway specified with this parameter. The Bfv API forwards the specified phone number to the gateway for routing purposes. When set, this parameter must contain an IP address in the form: `xxx.xxx.xxx.xxx:PortNumber` (Port number is optional) *Examples* `10.128.22.6:1720` (port number specified) `10.128.22.6` (no port number specified) **Note:** For the H.323 protocol, the port defaults to 1720 if not specified. *Range:* `0 – 255` for each dotted decimal position of the IP address. 1 – 65535 for the port number *Value Type:* dotted decimal *Default:* `0.0.0.0:0` **Note:** The Bfv API does not use this parameter if the configuration file specifies a value of 1 for the *h323_register* parameter (see *page 1261*).

h323_e164alias	Specifies the E.164 alias of the H.323 terminal. The system uses this alias during gatekeeper registration and call establishment. The alias identifies the phone number of the H.323 terminal.
	You can specify multiple aliases, each starting on a new line using the same parameter name. For example:
	`h323_e164alias 123456`
	`h323_e164alias 4084839648`
	`h323_e164alias 5102987468`
	`h323_e164alias 9627842899`
	When you specify multiple values, the system registers every value with the gatekeeper.
	Range: `1 – 128` characters (each)
	Value Type: character string restricted to numbers 0 through 9 and the star (*) and pound (#) symbols
	Default: `<blank>`

Table 24. Basic Predefined H.323 IP Call Control Stack Parameters (Continued)

Key Name	Description
h323_gatekeeper_id	Specifies the ID of the H.323 gatekeeper that the H.323 terminal expects to find during the gatekeeper discovery routine. If you do not set this parameter, the H.323 terminal attempts to register with the first gatekeeper it finds. `<blank>` Uses the first gatekeeper the H.323 terminal locates. *Value Type:* Unicode character string (up to 256 characters) *Default:* `<blank>`
h323_gatekeeper_ip_address *h323_gatekeeper_ip_address2* *h323_gatekeeper_ip_address3* *h323_gatekeeper_ip_address4* *h323_gatekeeper_ip_address5* *h323_gatekeeper_ip_address6*	Specifies the IP address of up to six H.323 gatekeepers that receive the registration request from the H.323 terminal. When set to the default value (0.0.0.0:0), the H.323 terminal performs a multicast gatekeeper discovery routine to find the gatekeeper using port number 1719. `xxx.xxx.xxx.xxx` Configures the system to use the specified H.323 gatekeeper. Set the gatekeeper IP address in the form: `xxx.xxx.xxx.xxx:PortNumber` `0.0.0.0:0` Configures the system to use a multicast process to discover the H.323 gatekeeper using port number 1719. *Range:* `0 – 255` for each dotted decimal position of the IP address. 1 – 65535 for the port number *Value Type:* dotted decimal *Default:* `0.0.0.0:0` (uses multicast discovery process and port number 1719)
h323_gatekeeper_ttl	Specifies the number of seconds to allow between registration request messages sent from the H.323 terminal to the gatekeeper. After reaching this limit, the H.323 terminal generates another registration request to the gatekeeper because the system now considers the previous request invalid. *Unit:* second *Range:* 0 – 32,000,000 (0 means that gatekeeper registrations do not expire; 32,000,000 seconds equals one calendar year) *Value Type:* integer *Default:* 0

Table 24. Basic Predefined H.323 IP Call Control Stack Parameters (Continued)

Key Name	Description

h323_h323IDalias	Specifies the H.323 ID of the H.323 terminal. The system uses this alias during gatekeeper registration and call establishment. The alias identifies the name of the H.323 terminal.
	You can specify multiple aliases, each starting on a new line using the same parameter name. For example:
	`h323_h323IDalias andrew`
	`h323_h323IDalias bob`
	`h323_h323IDalias charles`
	`h323_h323IDalias david`
	When you specify multiple values, the system registers every value with the gatekeeper.
	Range: Up to 256 characters (each)
	Value Type: Unicode character string
	Default: `<blank>`

h323_local_ip_address	Specifies the transport address of the H.323 terminal. The transport address can be an IP address or a combination of the IP address and the port number that the H.323 call control stack uses. When set to the default value (0.0.0.0:0), the system uses the IP address of the first Ethernet module in the system and port number 1720. Valid values are: `xxx.xxx.xxx.xxx` Configures the system to use the specified IP address for H.323 calls. Set the transport address in the form: `xxx.xxx.xxx.xxx:PortNumber` `0.0.0.0:0` Configures the system to use the address of the first Ethernet module and port number 1720. *Range:* `0 – 255` for each dotted decimal position of the IP address. 1 – 65535 for the port number *Value Type:* dotted decimal *Default:* `0.0.0.0:0` (uses first Ethernet module and port number `1720`)

Table 24. Basic Predefined H.323 IP Call Control Stack Parameters (Continued)

Key Name	Description
h323_manufacturerCode	Specifies a code that identifies the manufacturer of the H.323 terminal making the call. *Range:* 0 – 255 *Value Type:* decimal *Default:* 48 **Note:** 48 is the H.323 manufacturer code for Dialogic.
h323_Manufacturer	Specifies a string naming the H.323 terminal's manufacturer. *Range:* Up to 256 characters *Value Type:* Unicode character string *Default:* "Dialogic Corporation"
h323_max_sessions	Indicates the maximum number of concurrent H.323 calls that the host module can support at one time. Set this value to a number that at least doubles the number of channels in the system because the system can be tearing down a call while processing the next call. *Range:* `1` through `65535` (inclusive) *Value Type:* decimal *Default:* `256`
h323_register	Specifies an integer value that determines whether to register with an H.323 gatekeeper. `0` Does not register with an H.323 gatekeeper. `1` Registers with an H.323 gatekeeper. *Value Type:* integer *Default:* `0`

Table 24. Basic Predefined H.323 IP Call Control Stack Parameters (Continued)

Key Name	Description
h323_support_alternate_gk	Specifies whether to support alternate gatekeepers. The gatekeeper receiving the registration request from the H.323 terminal must also support alternate gatekeepers. When the H.323 terminal sends a registration request to its primary gatekeeper, the primary gatekeeper sends the H.323 terminal a list of alternate gatekeepers that it knows about. If, for some reason, the H.323 terminal can no longer communicate with its primary gatekeeper, it goes through this list of alternate gatekeepers and attempts to register with one of them. When the system does not support alternate gatekeepers and the H.323 terminal can no longer communicate with its primary gatekeeper, the H.323 terminal goes through the multicast gatekeeper discovery routine to find an available gatekeeper. `0` Does not support alternate gatekeepers. `1` Supports alternate gatekeepers when necessary. *Value Type:* integer *Default:* `0`
h323_t35CountryCode	Specifies a code identifying the international country that manufactured the H.323 terminal making the call. This code allows other H.323 terminals to know the origin of the H.323 terminal. The ITU-T Recommendation T35 Annex A lists the country codes used for H.323 Non-Standard Facilities (NSF). *Range:* 0 – 255 *Value Type:* decimal *Default:* 181 **Note:** 181 is the country code for USA.
h323_t35Extension	Specifies a modifier for the country code of the vendor's H.323 terminal making the call. *Range:* 0 – 255 *Value Type:* decimal *Default:* 0 **Note:** 0 is the extension for USA.

Table 25. Advanced Predefined H.323 IP Call Control Stack Parameters

Key Name	Description
h323_CalledPartyOption	Controls what fields in the H.225 SETUP message are returned as part of the called party number. The fields in SETUP used for called party number are: destination call signaling address, destination address, called party subaddress, called party extension (H.323 ID or E.164), called party number and destination extra call information. The following are the allowable parameter values: 0 Returns all H.225 SETUP fields described above if present. 1 Same as 0 except destination call signaling address is removed if present. 2 Only called party number is returned. *Value type:* integer *Default:* 1
h323_CallingPartyOption	Controls what fields in the H.225 SETUP message are returned as part of the calling party number. The following are the allowable parameter values: 0 Returns all H.225 SETUP fields. 1 Removes the transport address if present. 2 Only calling party phone number is returned if present. *Value type:* integer *Default:* 0
h323_h245Tunneling	Defines if H.323 tunneling is enabled or not. H.245 tunneling allows H.245 messages to be sent over the same IP address and port as H.225 messages. If tunneling is disabled, a new IP address and port specific for H.245 messages is established. The following are the allowable parameter values: 0 Disabled 1 Enabled *Value type:* integer *Default:* 1

Table 25. Advanced Predefined H.323 IP Call Control Stack Parameters

Key Name	Description
h323_FastStart	Determines outbound H.323 fast start call setup. The following are the allowable parameter values: 0 Outbound calls use H.323 slow start call setup. 1 Outbound calls use H.323 fast start call setup. *Value type:* integer *Default:* 1
h323_OlcRejectResponseTimeout	Controls how a master endpoint handles OLC conflict from a slave peer. The following are the allowable parameter values: -1 Causes master endpoint to send an OLC reject and wait for a peer slave to send non-conflicting OLC. 0 Causes the master endpoint to send RequestMode to peer slave. 1-1000 Duration (in ms) to wait after sending an OLC reject and before sending a RequestMode. If the master endpoint receives a non-conflicting OLC before the timeout period expires, RequestMode is not sent. *Value type:* integer *Default:* -1

Table 25. Advanced Predefined H.323 IP Call Control Stack Parameters

Key Name	Description
h323_h245Stage	The stage at which the local endpoint is allowed to transfer the H.245 address to the remote endpoint. This parameter is in effect when H.245 tunneling is disabled. Refer to the h323_h245Tunneling parameter. The following are the allowable parameter values: 0 — Earliest H.245 possible can send and act on addresses in all messages. 1 — Can send the address in the Call Proceeding message. 2 — Can send the address in only the Alerting message. 3 — Wait for the Connect message. 4 — Early H.245 send addresses in Setup and Connect messages only. 5 — No automatic sending of the address. 6 — No support for H.245 and the NoH245 Facility message is sent. *Value type:* integer *Default:* 5
h323_OverrideNumberingPlan	Manually overrides the numbering plan value in Q.931 called party number. The following are the allowable parameter values: -1 — Do not override. 0-15 — Numbering plan value. *Value type:* integer *Default:* -1

Table 25. Advanced Predefined H.323 IP Call Control Stack Parameters

Key Name	Description
h323_OverrideNumberingType	Manually overrides the numbering type value in Q.931 called party number. The following are the allowable parameter values: -1 Do not override. 0-7 Numbering type value. *Value type:* integer *Default:* -1
h323_RAS_Terminal_Type	Defines the terminal type sent as part of the RAS RRQ request. The following are the allowable parameter values: 0 Register as a terminal endpoint 1 Register as a gateway. *Value type:* integer *Default:* 0
h323_RAS_Voice_Supported_Prefixes	Defines a list of E.164 prefixes by which other endpoints may identify this endpoint. The parameter is used only when the h323_RAS_Terminal_Type filed is set to 1 - gateway. You can specify multiple values. The system registers each value with the gatekeeper. *Range:* `1 – 255` characters *Value Type:* character string (up to 256 characters) *Default:* `<blank>` (empty string)
h323_MediaWaitForConnect	Indicates that the recipient of the Setup message should not transmit media until sending the connect message. This setting controls the value of the Setup message mediaWaitForConnect flag. The following are the allowable parameter values: `TRUE` Recipients of the Setup message should not transmit media until sending the connect message. `FALSE` Recipients of the Setup message may transmit media without waiting to send the connect message. *Default:* `FALSE`

Table 26. Basic Predefined SIP IP Call Control Stack Parameters

Key Name	Description

sip_contact	Indicates the value provided in the SIP header for the *Contact* parameter. The *Contact* parameter contains a SIP uniform resource identifier (URI) or SIPS (secure SIP) URI that defines the address of the sender.
	When this parameter is set to its default value (`sip_Contact=0.0.0.0:0`), the SIP stack automatically attempts to find the IP address of the local host during initialization. If the host has not registered its host name, the SIP initialization process will fail and SIP calls will not be processed. To process SIP calls in this case, the value in the *sip_Contact* parameter must be specifically set to an IP address of one of the host network interface boards.
	When set, this parameter must contain an IP address in the form:
	`xxx.xxx.xxx.xxx:PortNumber` (port number is optional)
	`10.128.22.6:5060` (port number specified)
	`10.128.22.6` (no port number specified)
	This parameter can also be specified with an optional name prefix as shown in the following examples:
	`Name@xxx.xxx.xxx.xxx:PortNumber`
	`username@10.128.22.6:5060`
	`username@10.128.22.6`
	Note: For the SIP protocol, the port defaults to 5060 if not specified.
	Range: `0 – 255` for each dotted decimal position of the IP address.
	1 – 65535 for the port number
	Value Type: dotted decimal
	Default: `0.0.0.0:0` (system uses the IP address of the local host and port 5060)

sip_description_URI	Indicates the value used for the `u=` line in the SIP SDP. The `u=` line identifies the SIP uniform resource identifier (URI) of the session description. *Range:* `1 – 255` characters *Value Type:* character string (up to 256 characters) *Default:* `<blank>` (empty string)

Table 26. Basic Predefined SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_email	Indicates the value used for the `e=` line in the SIP SDP. The `e=` line identifies the email address of the person or entity responsible for the session. *Range:* `1 – 255` characters *Value Type:* character string (up to 256 characters) *Default:* `<blank>` (empty string)
sip_from	Indicates the value provided in the SIP header for the *From* parameter. The *From* parameter contains a display name and a SIP uniform resource identifier (URI) or SIPS (secure SIP) URI that identifies the originator of the session request. *Range:* `1 – 255` characters *Value Type:* character string (up to 256 characters) *Default:* `"Anonymous <sip:no_from_info@ anonymous.invalid>"`
sip_max_sessions	Indicates the maximum number of concurrent session initiation protocol (SIP) call control sessions. Set this value to a number that at least doubles the number of channels in the system because the system can be tearing down a call while processing the next call. *Range:* `1` through `1000` *Value Type:* decimal *Default:* `256`
sip_phone	Indicates the value used for the `p=` line in the SIP SDP. The `p=` line identifies the phone number to associate with the session. Phone numbers use the conventional international format: the number preceded by a + (plus symbol), the country code and a space or hyphen character. For example: `+1 408-370-0881` *Range:* `1 – 255` characters *Value Type:* character string (up to 256 characters) *Default:* `<blank>` (empty string)

Table 26. Basic Predefined SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_proxy_server1 *sip_proxy_server2* *sip_proxy_server3* *sip_proxy_server4*	Indicates the address (IPv4 or IPv6) of the specified SIP proxy server. The user can define a maximum of 4 proxy servers. `DHCP` Causes the system to use the SIP DNS server locator capability to discover the domain name of the SIP proxy server. Domain name Indicates the name or IP address of the proxy server. *Range:* 1 – 4 proxy servers specifying any valid domain name (for example, `www.my_sip_server.com`, `192.168.1.45, or [2000::201:1ef])` *Value Type:* character string (up to 256 characters) *Default:* `<blank>` (empty string indicating no proxy server defined) **Note:** Do not use the `DHCP` value. It is reserved for future use.
sip_registration_server1 *sip_registration_server2* *sip_registration_server3* *sip_registration_server4*	Indicates the address (IPv4 or IPv6) of the specified SIP registration server. The user can define a maximum of 4 registration servers. `DHCP` Causes the system to use the SIP DNS server locator capability to discover the domain name of the SIP registration server. Domain name Indicates the name or IP address of the registration server (up to 256 characters). *Range:* 1 – 4 registration servers specifying any valid domain name (for example, `www.my_sip_server.com`, `192.168.1.45, or [2000::201:1ef])` *Value Type:* character string (up to 256 characters) *Default:* `<blank>` (empty string indicating no registration server defined) **Note:** Do not use the `DHCP` value. It is reserved for future use.
sip_session_description	Indicates the value used for the `i=` line in the SIP SDP. The `i=` line provides a textual string that describes the session's purpose or provides information about the session. *Range:* `1 – 255` characters *Value Type:* character string (up to 256 characters) *Default:* `<blank>` (empty string)

Table 26. Basic Predefined SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_session_name	Indicates the value used for the `s=` line in the SIP SDP. The `s=` line provides a textual string that gives a name to the session. *Range:* `1 – 255` characters *Value Type:* character string (up to 256 characters) *Default:* `no_session_name`
sip_username	Indicates the name inserted into the `o=` line in the SIP session description protocol (SDP). The `o=` line defines the owner or creator of the session and the session identifier. This value must not contain spaces. Do not leave this field blank. Leaving it blank will cause calls to fail. – (dash) A dash or hyphen character indicates the absence of an owner name or session ID. *Range:* `1 – 255` characters *Value Type:* character string (up to 256 characters) *Default:* – (dash or hyphen character)
sip_user_agent	Specifies a text description of the software/hardware/product provided in the User-Agent header. The syntax is specified in RFC 3261. *Default:* `Brktsip/(sdk version> (Dialogic)`

Table 27. Advanced SIP IP Call Control Stack Parameters

Key Name	Description

sip_ContactV6	Indicates the IPv6 IP address value provided in the SIP header for the Contact parameter. The Contact parameter contains a SIP uniform resource identifier (URI) or SIPS (secure SIP) URI that defines the address of the sender.
	If a value is specified for the sip_ip_interfaceV6 key name, this parameter is ignored. If the value of this parameter and the sip_ip_interfaceV6 parameter are blank, this is an invalid and unsupported configuration.
	When set, this parameter must contain an IPv6 address in the form:
	`[<IPv6 Address>%<Scope ID>]:PortNumber`
	Where:
	■ IPv6 addresses must be enclosed in brackets
	■ Link-Local IPv6 addresses must have their Scope ID specified after the IPv6 address and be separated by a % character
	■ Port number is optional
	`[2000::2ef3:1dff:ea3]:5060`
	(Global IPv6 address, no Scope ID or port number specified)
	`[fe80::1f4:189c:74da:69f7%3]`
	(Link-Local IPv6 address with Scope ID, no port number specified)
	Note: For the SIP protocol, the port defaults to 5060 if not specified.
	This parameter can also be specified with an optional name prefix as shown in the following examples:
	`Name@[IPv6 Address]:PortNumber`
	`username@[2000::2ef3:1dff:ea3]:5060`
	`username@[fe80::1f4:189c:74da:69f7%3]`
	Range: 1 – 65535 for the port number
	Value Type: character string (up to 256 characters)
	Default: <blank>

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description

sip_default_gateway
sip_gateway2
sip_gateway3
sip_gateway4
	Indicates the IP address (IPv4 or IPv6) of a SIP gateway to use for outbound calls. If a user only specifies a phone number when making a SIP call and the application is not using a SIP proxy server, the Bfv API forwards the call to one of the SIP gateways specified here. The user can define a maximum of 4 SIP gateways. The Bfv API forwards the specified phone number to the gateway for routing purposes.
	When set, this parameter must contain an IP address in the form:
	IPv4
	`xxx.xxx.xxx.xxx:PortNumber (port number is optional)`
	`10.128.22.6:5060 (port number specified)`
	`10.128.22.6 (no port number specified)`
	IPv6
	`[<IPv6 Address>]:PortNumber (port number is optional)`
	`[2000::2ef3:1dff:ea3]:5060 (port number specified)`
	`[2000::2ef3:1dff:ea3] (no port number specified)`
	IPv6 addresses must be enclosed within brackets.
	Note: For the SIP protocol, the port defaults to 5060 if not specified.
	Range: `0 – 255` for each dotted decimal position of the IP address.
	1 – 65535 for the port number
	Value Type: dotted decimal
	Default: `0.0.0.0:0` (no default gateway defined)
	When multiple SIP gateways are configured, SIP OPTIONS requests will be periodically transmitted to each of the gateways in order to determine their operational status. SIP gateways that fail to respond to a SIP OPTIONS request or respond with either a 503 (Service Unavailable) or 505 (SIP Version Not Supported) will be determined to be DOWN and unavailable to process SIP calls. All other SIP responses will result in a SIP gateway to be considered to be UP and available to process SIP calls.
	Outbound SIP gateway calls will always be routed to the highest priority SIP gateway whose status is UP with *sip_default_gateway* being the highest priority SIP gateway and *sip_gateway4* being the lowest priority SIP gateway.
	Note: The Bfv API does not use these parameters if the configuration file specifies a *sip_registration_server* or *sip_proxy_server* (see *page 1269*).

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_options_up_interval	Specifies the interval that SIP OPTIONS requests should be transmitted to SIP gateways whose status is currently UP. By periodically transmitting SIP OPTIONS requests to a SIP gateway and keeping track of whether or not responses are received and the type of responses, the operational status of SIP gateways can be determined. *Range:* 60 – 3600 seconds *Value Type:* Unsigned integer *Default:* 120
sip_options_down_interval	Specifies the interval that SIP OPTIONS requests should be transmitted to SIP gateways whose status is currently DOWN. By periodically transmitting SIP OPTIONS requests to a SIP gateway and keeping track of whether or not responses are received and the type of responses, the operational status of a SIP gateways can be determined. *Range:* 30 – 3600 seconds *Value Type:* Unsigned integer *Default:* 60

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description

sip_ip_interface	Specifies the identity of the device on the PC with the IP interface that the SIP Call Control stack can use for sending/receiving SIP messages to/from from an IPv4 IP address.
	Set the value of this parameter to the name of any device in the PC with an IP interface. If you do not provide a value (blank string), the virtual module chooses the first interface in the PC to send its messages.
	Note: The format for the value provided by this parameter is operating system dependent.
	The Windows format for the value provided in this parameter is:
	■ The name of the IP device (Global Unique IDentifier (GUID)) followed by
	■ A colon (:) character followed by
	■ The index number IPv4 IP address on the device
	For example:
	`{4D36E96E-E325-11CE-BFC1-08002BE10318}:0`
	The Linux format is the ethernet device name.
	For example:
	`sip_ip_interface=eth0`
	Value Type: character string (up to 256 characters)
	Default: <blank> (the virtual module uses the first interface in the PC for sending SIP messages)
	If the default (i.e. <blank>) value is specified for this parameter, the SIP Call Control stack will use the value specified for the sip_contact parameter.

sip_ip_interface_port	Specifies the IPv4 port that should be used for SIP call control. This parameter is used when the parameter IPv4 Interface For SIP [sip_ip_interface] has been set. *Range:* 1 – 65535 for the port number *Value Type:* integer *Default:* 5060

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_ip_interfaceV6	Specifies the identity of the device on the PC with the IP interface that the SIP Call Control stack can use for sending/receiving SIP messages to/from an IPv6 IP address. Set the value of this parameter to the name of any device in the PC with an IP interface. If you do not provide a value (blank string), the virtual module chooses the first interface in the PC to send its messages. **Note:** The format for the value provided by this parameter is operating system dependent. The Windows format for the value provided in this parameter is: ■ The name of the IP device (Global Unique IDentifier (GUID)) followed by ■ "A colon (:) character followed by ■ "The index number of the IPv4 IP address on the device For example: **{4D36E96E-E325-11CE-BFC1-08002BE10318}:0** The Linux format is the ethernet device name. For example: **sip_ip_interfaceV6=eth0** *Value Type:* character string (up to 256 characters) *Default:* <blank> If the default (i.e. <blank>) value is specified for this parameter, the SIP Call Control stack will use the value specified for the sip_contactV6 parameter. **Note:** If the value of this parameter and the sip_contactV6 parameter are blank, this is an invalid and unsupported configuration.

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_ip_preference	Specifies the IP family preference that should be used for SIP call control. The following are the allowable parameter values: ipv4_only — Only IPv4 supported for SIP calls ipv6_only — Only IPv6 supported for SIP calls ipv4_preferred — IPv4 and IPv6 supported for SIP calls. If both IPv4 and IPv6 addresses resolved for destination of outbound SIP call upon DNS lookup, IPv4 will be selected. ipv6_preferred — IPv4 and IPv6 supported for SIP calls. If both IPv4 and IPv6 addresses resolved for destination of outbound SIP call upon DNS lookup, IPv6 will be selected. *Value Type*: character string *Default*: ipv4_only
sip_ip_interface_portV6	Specifies the IPv6 port that should be used for SIP call control. This parameter is used when the parameter IPv6 Interface For SIP [sip_ip_interfaceV6] has been set. *Range*: 1 – 65535 for the port number *Value Type*: integer *Default*: 5060
sip_max-forwards	Indicates the value provided in the SIP header for the *Max-Forwards* parameter. The value in the *Max-Forwards* parameter serves to prevent loops by limiting the number of hops a SIP request can make on the way to its destination. The value consists of an integer that the system decrements by one at each hop. When this value reaches 0, the system discards the request. *Range:* 1 – 200 inclusive *Value Type:* decimal *Default:* 70

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_registration_interval	Indicates the frequency for sending REGISTER requests to a registration server. `0` Indicates that the system does not send REGISTER requests to the registration server. *Unit:* minutes *Range:* `1` to `65535` inclusive *Value Type:* decimal *Default:* `60`
sip_registration_ server<n>_aor	Indicates the address or record (aor) SIP uniform resource identifier (URI) that is bound to the *sip_Contact* (see *page 1267*). Currently, the SIP host module only allows one contact for each address of record. The address of record is set in the *To:* and *From:* fields in the SIP REGISTER message. Set the *n* value to the applicable number of the registration server. *Range:* `1 – 255` characters *Value Type:* character string (up to 256 characters) *Default:* `<blank>` (empty string indicating no username defined)
sip_registration_ server<n>_expires	Indicates a value in seconds that defines how long the address of record or contact remains valid in the registration server. Set the *n* value to the applicable number of the registration server. *Unit:* seconds *Range:* `1` to `1,000,000` inclusive *Value Type:* decimal *Default:* `3600`
sip_registration_ server<n>_password	Indicates the password used as part of the authentication process when the registration servers require authentication. Authentication is based on RFC2617. Set the *n* value to the applicable number of the registration server. *Range:* `1 – 255` characters *Value Type:* character string (up to 256 characters) *Default:* `<blank>` (empty string indicating that the password is `NULL`)

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_registration_ server<n>_username	Indicates the username used as part of the authentication process when the registration servers require authentication. Authentication is based on RFC2617. Set the *n* value to the applicable number of the registration server. *Range:* 1 – 255 characters *Value Type:* character string (up to 256 characters) *Default:* <blank> (empty string indicating no username defined)
sip_registration_ proxied	This parameter selects if the SIP REGISTER will be sent to the active SIP Proxy Server. The following are the allowable parameter values: *True*: Send all REGISTER mesages via the configuration SIP Proxy Server *False*: Send all REGISTER messages directly to the SIP Register
sip_reject_call_not_answered	This parameter specifies the SIP response code to transmit when an inbound SIP call is received and a call is not answered by the application. This can occur when all channels are busy, when a call is completed but the application has not restarted ring detection, or when the Boston Host Service has been started and a call arrives before the application is started. *Range:* 400 to 699 *Value Type:* Unsigned integer *Default:* 486 (Busy Here)
sip_reject_unsupported_media	This parameter specifies the SIP response code to transmit when an inbound SIP INVITE is received and the SDP does not contain any supported media types. *Range:* 400 to 699 *Value Type:* Unsigned integer *Default:* 488 (Not Acceptable)

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_reject_t38_renegotiation	This parameter specifies the SIP response code to transmit when an inbound SIP INVITE is received that specifies a codec of T.38 in the SDP and the Fax Transporting Protocol parameter [fax_transport_protocol] is set to G.711 pass-through only (that is, t38_never). *Range:* `400 to 699` *Value Type:* `Unsigned integer` *Default:* `488 (Not Acceptable)`
sip_route	Indicates the value provided in the SIP header for the *Route* parameter. The *Route* parameter contains the address or multiple addresses of proxy servers. The system uses the *Route* parameter to force routing of a request through the listed set of proxies. *Range:* `1 – 255` characters (multiple addresses separated by a comma `","`) *Value Type:* character string (up to 256 characters) *Default:* `<blank>` (empty string indicating no forced routing)
sip_session_timer_minse	Establishes the lower bound for the session refresh interval in seconds. The Min-SE header is not normally present in the sent request (except all requests following a 422 response. Additionally, if this parameter is set to any other value other than the default, the Min-SE header will appear in all sent requests. To prevent the exchange of an excessive number of session refresh messages, the minimum non-zero value allowed for this parameter is 90. *Range:* -1 to 65535 seconds. Values of 0 to 89 will use 90 *Value Type:* Signed integer *Default:* `-1 (do not send Min-SE header)`
sip_session_timer_ refresh_method	Defines whether the refresh method uses the INVITE or UPDATE transaction to refresh the session. The session must be identified as the refresher side to send the refresh transaction; otherwise, the remote session is responsible for session refresh. *Range:* 0 - Use the INVITE method 1 - Use the UPDATE method *Value Type:* Unsigned integer *Default:* 0 - Use the INVITE method

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_session_timer_ session_expires	Defines the maximum time in seconds before a session is considered timed out without a successful INVITE or OPTIONS transaction. *Range:* 0 – 65535 *Value Type:* Signed integer *Default:* 0 - Session Timer Disabled
sip_redirect_as_calling_ party	Specifies whether or not to report the redirect number as the calling party number to the application. Set the parameter as follows: *Range:* 0 - Causes the system to use the original calling party number as the number reported to the application. 1 - Causes the system to use the redirect number as the calling party number reported to the application. Selecting this option removes any association between the original calling party number and the call. *Value Type:* Unsigned integer *Default:* 0 for backward compatibility
sip_redirect_as_called_party	Specifies whether or not to report the redirect number as the called party number to the application. Set the parameter as follows: *Range:* 0 - Causes the system to use the original called party number as the number reported to the application. 1 - Causes the system to use the redirect number as the called party number reported to the application. Selecting this option removes any association between the original called party number and the call. *Value Type:* Unsigned integer *Default:* 0 for backward compatibility

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_RFC3325_Identity	Inserts an Identity header and Privacy header with the identity of the call originator specified in the args_cc.calling_party. The value of the Identity header will be authenticated by the application when required. *Range:* 0 - No RFC 3325 Identity or Privacy headers will be inserted. 1 - Insert the header Privacy=noe and P-Asserted-Identity with the calling_party for outbound calls. 2 - Insert the header Privacy=none and P-Preferred_Identity with the calling_party for outbound calls. *Value Type:* Unsigned integer *Default:* 0 - No RFC 3325 Identity or Privacy headers will be inserted.
sip_tcp_enable	Indicates that support for the Transmission Control Protocol (TCP) transport protocol should be enabled for SIP call processing. Note that setting this parameter value to TRUE enables simultaneous support for receiving SIP calls using either the TCP or User Datagram Protocol (UDP) transport protocols. For outbound SIP calls, when this parameter is set to TRUE, the transport protocol used (either UDP or TCP) for the SIP call control messages is determined by the value of the ***sip_ transport_protocol*** call control configuration file parameter or can be specified at runtime via the ***call_transport*** field in either the ***args_cc*** or ***args_telephone*** structures, as appropriate to your application. The following are the allowable parameter values: TRUE Enable TCP transport protocol functionality for the SIP stack FALSE Disable TCP transport protocol functionality for the SIP stack *Default:* FALSE

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_transport_protocol	Indicates the transport protocol, either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) to use for outbound SIP calls. The following are the allowable parameter values: UPD Use UDP transport protocol for outbound SIP calls. TCP Use TCP transport protocol for outbound SIP calls. TLS Use TLS transport protocol for outbound SIP calls. *Default:* UDP *NOTE:* This parameter specifies the default transport protocol to use for outbound SIP calls. However, if a value is specified at runtime via the ***call_transport*** field in either the ***args_cc*** or ***args_telephone*** structures, as appropriate to your application, that value will take precedence over the value specified for this parameter.
sip_T1_timeout	Specifies the SIP T1 timeout and is an estimate of the Round Trip Time (RTT) of transactions between a client and server. For example, a SIP Client will attempt to send a request to a SIP Server. The time it takes between sending out the request to the point of getting a response is the SIP T1 timer. If no response is received the timeout is increased to (2*T1) and then (4*T1) doubling the previous timeout each time the SIP request is retransmitted. *Unit:* milliseconds *Range:* 100 – 60000 *Value Type:* decimal *Default:* 500
sip_max_invite_retransmissions	Specifies the maximum number of SIP INVITE request transmissions. A call will terminate after the INVITE has been transmitted this many times. If no response is received after T1 Timeout (***sip_T1_timeout***), the INVITE will be retransmitted at an interval of double the previous timeout. *Range:* 1 – 255 *Value Type:* integer *Default:* 7

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
sip_RFC6913_enable	Indicates that support for RFC 6913 (Indicating Fax over IP Capability in the Session Initiation Protocol (SIP)) should be enabled for SIP call processing. When this parameter is set to TRUE, outbound SIP REGISTER requests will include a "sip.fax" media feature tag appended to the SIP Contact header. The value of the "sip.fax" media feature tag transmitted in SIP REGISTER messages will be determined by the value of the ***fax_transport_protocol*** parameter in the *t38parameters* section of the *callctrl.cfg* file as follows: *fax_transport_protocol value* — *"sip.fax" value* t38_never — passthrough t38_only — t38 t38_first — t38 Not specified in callctrl.cfg file — t38 In addition to this, when this parameter is set to TRUE, outbound SIP INVITE requests will include a "sip.fax" media feature tag in an Accept-Contact header. The value of the "sip.fax" media feature tag in transmitted SIP INVITE requests is determined by the value of the ***fax_transport_protocol*** call control configuration file parameter as noted in the table above or can be specified at runtime via the ***fax_media_feature_tag*** field in either the ***args_cc*** or ***args_telephone*** structures, as appropriate to your application. The following are the allowable parameter values: TRUE — Enable RFC 6913 functionality for the SIP stack. FALSE — Disable RFC 6913 functionality for the SIP stack. *Default:* FALSE
srtp_enabled	Indicates SR140 support for SDES SRTP. This feature requires a valid security license to be installed on the system. When set to TRUE, the ***srtp_config_filename*** must point to a valid SRTP configuration file. The following are the allowable parameter values: TRUE — Enable SDES SRTP support for the SR140. FALSE — Disable SDES SRTP support for the SR140. *Default:* FALSE

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
srtp_config_filename	Full path and filename of the SRTP configuration parameters file. This file contains the SRTP configuration parameters used by the SR140 when SRTP is enabled. If SRTP is enabled and the configuration file cannot be found, an error will be generated in the ECC log. FULLPATH Contains the full path to the SRTP configuration file. *Value Type:* character string *Default:* srtp.cfg
fips_enable	Indicates SR140 support for using the FIPS Object Module. The FIPS Object Module was designed and implemented to meet FIPS 140-2 requirements. The FIPS publication standards are available at: *http://csrc.nist.gov/publications/PubsFIPS.html*. The following are the allowable parameter values: TRUE Enable FIPS object module. FALSE Disable FIPS object module. *Default:* FALSE
sip_tls_enabled	Indicates SR140 support for TLS. This feature requires a security license. When enabled, the ***sip_tcp_enable*** parameter must be set to TRUE. When enabled, requires ***tls_config_filename*** to be set to a valide filename. The following are the allowable parameter values: TRUE Enable TLS support for the SR140. FALSE Disable TLS support for the SR140. *Default:* FALSE
tls_config_filename	Full path and filename of the TLS configuration parameters file. This file contains the TLS configuration parameters used by the SR140 when TLS is enabled. FULLPATH Contains the full path to the TLS configuration file. *Value Type:* character string *Default:* siptls.cfg
sip_tls_port	Specifies the port that should be used for SIP TLS call control. *Range:* 1 - 65535 for the port number *Value Type:* integer *Default:* 5061

Table 27. Advanced SIP IP Call Control Stack Parameters (Continued)

Key Name	Description
block_udp_port	When TLS is enabled, this flag is used to indicate if the SR140 should block the usage of UDP. The following are the allowable parameter values: TRUE UDP port blocked. FALSE UDP port allowed. *Default:* TRUE
block_tcp_port	When TLS is enabled, this flag is used to indicate if the SR140 should block the usage of TCP. The following are the allowable parameter values: TRUE TCP port blocked. FALSE TCP port allowed. *Default:* TRUE

Configuring Ethernet Interface Parameters

In the module.#/`ethernet.#` section of the configuration file, identify the module's interface `x` using the 1-based index of the Ethernet interface. This index allows an application to configure modules with multiple interfaces. Set the following parameters for a module configured to use an Ethernet interface.

Parameter	Value
dhcp	Specifies whether the Ethernet interface uses dynamic host configuration protocol (DHCP) to request an IP address. Set this parameter to: `DISABLED` Indicates that the *ip_address* parameter provides the IP address for the port. `ENABLED` Configures the Ethernet interface to use DHCP to request an IP address. *Value Type:* character string *Default:* `DISABLED` **Note:** This parameter is ignored and reserved for future use.
ethernet_speed	Specifies the speed of the module's Ethernet interface. Set this parameter to: `AUTO` Configures the interface to automatically sense the speed of the network. `10` Sets the speed of the interface to 10 Mbps. `100` Sets the speed of the interface to 100 Mbps. *Unit:* Mbps *Range:* `10`, `100`, or `AUTO` *Value Type:* character string *Default:* `AUTO`

Parameter	Value
ip_address	Specifies the IPv4 IP address of the module's Ethernet interface. Set this parameter only if you set the value in the *dhcp* parameter to `DISABLED`. `xxx.xxx.xxx.xxx` Configures the Ethernet interface to use the specified IP address. *Value Type:* dotted decimal *Default:* None **Note:** The Dialogic® Brooktrout® module does not support the domain naming system (DNS) data base. Your application has the responsibility of converting domain names into resolved dotted-decimal notation IP addresses.
ip_addressV6	Specifies the IPv6 IP address of the module's Ethernet interface. `[<IPv6 Address>%<Scope ID>]` Where: ■ IPv6 addresses must be enclosed in brackets. ■ "Link-Local IPv6 addresses must have their Scope ID specified after the IPv6 address separated by a % character *Value Type*: character string (up to 256 characters) *Default*: None **Note:** The Dialogic® Brooktrout® module does not support the domain naming system (DNS) data base. Your application has the responsibility of converting domain names into resolved IP addresses.
ip_arp_timeout	Specifies the arp (address resolution protocol) timeout value that the module's Ethernet interface uses. Set this parameter to: *Unit:* minutes *Range:* 0 – 1,000,000 where 0 indicates that the timeout is disabled. *Value Type:* decimal *Default:* `10`
ip_broadcast	Specifies the IP broadcast address of the module's Ethernet interface. Set this parameter to: `xxx.xxx.xxx.xxx` Configures the Ethernet interface to use the specified broadcast address. *Value Type:* dotted decimal *Default:* None

Parameter	Value
ip_gateway	Specifies the gateway address of the module's Ethernet interface. Set this parameter to: `xxx.xxx.xxx.xxx` Configures the Ethernet interface to use the specified gateway address. *Value Type:* dotted decimal *Default:* None

ip_interface	Specifies the identity of the device on the PC with the IP interface that the virtual module can use for sending/receiving IP messages to/from an IPv4 IP address.
	Note: This parameter only applies to host-based fax applications using a virtual module.
	Set the value of this parameter to the name of any device in the PC with an IP interface. If you do not provide a value (blank string), the virtual module will use the value specified for the ip_address keyword. If no value is specified for either keyword, the first available IPv4 address on the PC will be selected for the PC to send its messages.
	Note: The format for the value provided by this parameter is operating system dependent.
	The Windows format for the value provided in this parameter is:
	■ The name of the IP device (Global Unique IDentifier (GUID)) followed by
	■ A colon (:) character followed by
	■ The index number of the device's IP address
	For example:
	`{4D36E96E-E325-11CE-BFC1-08002BE10318}:0`
	The Linux format is the ethernet device name.
	For example:
	`ip_interface=eth0`
	Value Type: character string (up to 256 characters)
	Default: `<blank>` (the virtual module uses the first interface in the PC for sending IP messages)

Parameter	Value
ip_interfaceV6	Specifies the identity of the device on the PC with the IP interface that the virtual module can use for sending/receiving IP messages to/from an IPv6 IP address. Set the value of this parameter to the name of any device in the PC with an IP interface. If you do not provide a value (blank string), the virtual module will use the value specified for the ip_addressV6 keyword. If no value is specified for either keyword, the first available IPv6 address on the PC will be selected for the PC to send its messages. **Note:** The format for the value provided by this parameter is operating system dependent. The Windows format for the value provided in this parameter is: ■ The name of the IP device (Global Unique IDentifier (GUID)) followed by ■ A colon (:) character followed by ■ The index number of the IPv6 IP address on the device For example: {4D36E96E-E325-11CE-BFC1-08002BE10318}:0 The Linux format is the ethernet device name. For example: `ip_interfaceV6=eth0` *Value Type*: character string (up to 256 characters) *Default*: <blank> If the default (i.e. <blank>) value is specified for this parameter, the virtual Module will use the value specified for the ip_addressV6 parameter.
ip_netmask	Specifies the netmask address of the module's Ethernet interface. Set this parameter only if you set the value in the *dhcp* parameter to `DISABLED`. `xxx.xxx.xxx.xxx` Configures the Ethernet interface to use the specified netmask address. *Value Type:* dotted decimal *Default:* `0.0.0.0`

Parameter	Value
ip_preference	Specifies the IP family preference that should be used by the virtual module for sending IP messages. The following are the allowable parameter values: ipv4_only — Only IPv4 supported ipv6_only — Only IPv6 supported ipv4_preferred — IPv4 and IPv6 both supported. For outbound SIP calls, the specific IP family type used for the IP messages sent by the virtual module will be determined by the SIP Call Control stack. ipv6_preferred — IPv4 and IPv6 both supported. For outbound SIP calls, the specific IP family type used for the IP messages sent by the virtual module will be determined by the SIP Call Control stack. *Value Type*: character string *Default*: ipv4_only
media_port_max	Specifies the highest IP port number that the module can use. Set this value to a value 1000 above the value specified for the *media_port_min* parameter. `57000` Sets this value as the highest port number. *Range:* `2024 – 65535` *Value Type:* decimal *Default:* `57000`
media_port_min	Specifies the lowest IP port number that the module can use for media transmissions. Set this value to a value 1000 below the value specified for the *media_port_max* parameter. `56000` Sets this value as the lowest port number. *Range:* `1024 – 64535` *Value Type:* decimal *Default:* `56000`
t38_offer_as_ced	Specifies whether to generate a CED detected event when receiving a T.38 offer. A T.38 offer is a SIP re-Invite or H.323 requestMode message indicating an IP endpoint wishes to switch the IP call to T.38. This allows applications performing call progress to detect the T.38 offer and transition to fax. `false` - Don't send CED tone detected. `true` - Send CED tone detected. *Value Type*: Boolean *Default:* TRUE

Configuring A Module To Use An IP Call Control Stack

In the `module.#/host_cc.#` section of the configuration file, identify each IP call control stack that a module can use. You can:

- Specify the same stack for more than one module.
- Configure each module to support a maximum of 9 different stacks.

Add one `module.#/host_cc.#` section for each IP call control stack the module can use, starting at 1 (`host_cc.1` through `host_cc.9`).

Set the following parameters for each `host_cc.#` section of the call control configuration file.

Parameter	Value
host_module	Specifies the number that identifies the IP call control stack that the module can use. Set this parameter to match the `host_module` number identifier (see *page 1246*) associated with the IP call control stack the module can use. Valid values are: *Range:* `1 – 9` *Value Type:* decimal *Default:* `1`
number_of_channels	Specifies the number of channels enabled to use the specified stack. This number must not exceed the number of available channels on the module. The Bfv API allocates the first available channels on the module to this stack. When the module can use multiple stacks, the Bfv API maps the channels to the stacks in the order that the stacks appear in the configuration file. If you configure a module to use telephony ports and an IP call control stack, the Bfv API allocates channels to the telephony interface first. *Range:* `1 – 1024` (not to exceed the maximum number of available channels on the module) *Value Type:* decimal *Default:* `1`

Configuring A Module To Use An RTP Stack

host_module.#/rtp

The `host_ module.#/rtp` section provides parameters to configure an RTP stack on the SR140 virtual modules and IP-enabled boards. If a value provided for a parameter is out of range, the parameter's default value will be used instead.

Parameter	Value
rtp_frame_duration	Specifies the duration of outbound RTP packets in multiple of 10ms. SR140 virtual modules do not support outbound 10ms packets. *Unit*: ms *Range:* 10 - 30 *Value Type*: decimal *Default*: 20
rtp_voice_frame_replacement	Specifies how to treat missing inbound voice frames. 0 Missing frame is replaced with silence. 1 Missing frame is replaced with the previous frame. *Value Type:* decimal *Default:* 0 (Silence)
rtp_jitter_buffer_depth	Specifies the depth of the RTP jitter buffer in multiples of 10ms. The depth of the jitter buffer determines the playout delay imposed by the RTP stack before passing data to the signal processing algorithms. This delay should be considered when performing time sensitive tone detection in IVR applications. A longer playout delay also provides a better opportunity for recovering from RTP packets received out of order. *Unit*: ms *Range*: 0 - 500 *Value Type*: decimal *Default*: 100
rtp_silence_control	Determines how silence, in an outbound RTP stream, is treated. inband - RTP silence is not suppressed suppress - RTP silence is suppressed *Value Type:* character string *Default:* inband

Parameter	Value
rtp_type_of_service	Determines how the first six bits of the ToS DCSP (Differentiated Services Point Code) field in the IP header are set for RTP packets. This parameter is available for Linux only. For Windows, the value of the DSCP bits is set from a group policy. *Unit*: none *Range:* 0 - 63 *Value Type:* decimal *Default:* 0
rtp_codec	Defines the codecs supported and codec order offered to a remote device during call negotiation. This parameter may be set to one or two codecs. Codec names should be entered without quotes and separated by a space. The first codec after the keyword is given the highest order of priority. For example, rtp_codec=pcmu pcma will offer both codecs, but pcmu will be the preferred one. Another option is to set rtp_codec=pcmu followed by rtp_code=pcma on the next line. This will offer both codecs, but pcmu will be the preferred one. *Unit*: none *Range:* pcmu (PCM Mu-law), pcma (PCM A-law) *Value Type:* null-terminated case insensitive string *Default:* pcmu
rtp_timing_control	Specifies whether to enable an alternate interval timing between RTP packets. A value of false will not change the interval timing between RTP packets. A value of true will enable an alternate interval timing between RTP packets. This parameter is only valid for the SR140 under Windows operating systems. It should not be used in a system using only TR1034 IP boards. This parameter should only be used when directed to do so by Dialogic Technical Services and Support. *Value Type:* Boolean *Default:* FALSE

Parameter	Value
rtp_type_of_service	Determines how the first six bits of the ToS DCSP (Differentiated Services Point Code) field in the IP header are set for RTP packets. This parameter is available for Linux only. For Windows, the value of the DSCP bits is set from a group policy. *Unit*: none *Range:* 0 - 63 *Value Type:* decimal *Default:* 0
rtp_codec	Defines the codecs supported and codec order offered to a remote device during call negotiation. This parameter may be set to one or two codecs. Codec names should be entered without quotes and separated by a space. The first codec after the keyword is given the highest order of priority. For example, rtp_codec=pcmu pcma will offer both codecs, but pcmu will be the preferred one. Another option is to set rtp_codec=pcmu followed by rtp_code=pcma on the next line. This will offer both codecs, but pcmu will be the preferred one. *Unit*: none *Range:* pcmu (PCM Mu-law), pcma (PCM A-law) *Value Type:* null-terminated case insensitive string *Default:* pcmu
rtp_timing_control	Specifies whether to enable an alternate interval timing between RTP packets. A value of false will not change the interval timing between RTP packets. A value of true will enable an alternate interval timing between RTP packets. This parameter is only valid for the SR140 under Windows operating systems. It should not be used in a system using only TR1034 IP boards. This parameter should only be used when directed to do so by Dialogic Technical Services and Support. *Value Type:* Boolean *Default:* FALSE

Parameter	Value

rfc_2833_outgoing_advertise	Advertises RFC 2833 support by adding the lines shown below to the SDP body of an outgoing SIP INVITE message. It does not enable detection of RFC 2888 DTMF RTP events. This parameter's setting is ignored when *rfc_2833_enabled* is TRUE.
	
	This parameter is only valid for the SR140.
	
	This parameter should only be enabled when advised by Dialogic Technical Services and Support.
	
	Value Type: Boolean
	Default: FALSE
	
	Lines added to SDP body:
	`m=audio xxxxx RTP/AVP 101`
	`a=rtpmap:101 telephone-event/8000`
	`a=fmtp:101 0-15`
	
	Where “xxxxx” is the local IP port.

rfc_2833_enabled	Enables detection, but not generation, of RFC 2888 DTMF Digits as RTP events. It adds the SDP lines, as shown in *rfc_2833_outgoing_advertise*, to the SDP body of an outgoing SIP INVITE message or to a 200 OK response message when the incoming INVITE contained a telephone-event parameter. This parameter is only valid for the SR140. *Value Type:* Boolean *Default:* FALSE

Examples of PSTN Call Control (callctrl.cfg) Files

This section provides example settings in call control configuration files for the following port-specific configurations:

- Analog DID ports — see *page 1297*
- Analog loop start ports — see *page 1298*
- ISDN BRI ports — see *page 1300*
- E1 ISDN ports — see *page 1301*
- E1 R2 CAS ports — see *page 1302*
- T1 ISDN ports — see *page 1303*
- T1 QSIG ports — see *page 1304*
- T1 Robbed Bit Signaling ports — see *page 1306*

To use one of the sample call control configuration files, select the file that most closely matches the desired configuration of your system.

Note: If a parameter specifies a file name, specify the full path to the file.
For example, if the install location is C:Brooktrout/Boston then specify the following parameter as:

```
protocol_file=C:/Brooktrout/Boston/config/
analog_loopstart_us.lec
```

Refer to *Sample Configuration Files on page 1318* for a listing of sample configuration files included with the Brooktrout SDK that provide a good starting point for your configuration. You might need to customize them for your system.

Analog DID Port-Specific Configuration File Example

The following code shows an example configuration file for ports using analog DID interfaces. The code specifies values for module number zero that automatically configure all the modules in the system to the settings in the file.

```
l314_trace=none
l413_trace=none
api_trace=none
trace_file=ecc.log
[module.0]
   pcm_law=mulaw
[module.0/port.1]
   port_config=analog_did
   line_coef=0
   protocol_file=winkstart.lec
   reject_incomplete_did=false
   max_did_digits=0
   did_offset=0
   did_timeout=5
[module.0/port.2]
   port_config=analog_did
   line_coef=0
   protocol_file=winkstart.lec
   reject_incomplete_did=false
   max_did_digits=0
   did_offset=0
   did_timeout=5
[module.0/port.3]
   port_config=analog_did
   line_coef=0
   protocol_file=winkstart.lec
   reject_incomplete_did=false
   max_did_digits=0
   did_offset=0
   did_timeout=5
[module.0/port.4]
   port_config=analog_did
   line_coef=0
   protocol_file=winkstart.lec
   reject_incomplete_did=false
   max_did_digits=0
   did_offset=0
   did_timeout=5
```

Analog Loop Start Port-Specific Configuration File Example

The following code shows an example configuration file for ports using analog loop start interfaces. The code specifies values for module number zero that automatically configure all the modules in the system to the settings in the file.

```
l3l4_trace=basic
l4l3_trace=basic
api_trace=verbose
internal_trace=verbose
trace_file=call_control_trace.txt
max_trace_file_size=10
[module.0]
    pcm_law=mulaw
[module.0/port.1]
    port_config=analog
    protocol_file=analog_loopstart_us.lec
    num_rings=2
    input_gain=0
    output_gain=0
    caller_id=enabled
    country=us600.qslac
[module.0/port.2]
    port_config=analog
    protocol_file=analog_loopstart_us.lec
    num_rings=2
    input_gain=0
    output_gain=0
    caller_id=enabled
    country=us600.qslac
[module.0/port.3]
    port_config=analog
    protocol_file=analog_loopstart_us.lec
    num_rings=2
    input_gain=0
    output_gain=0
    caller_id=enabled
    country=us600.qslac
```

```
[module.0/port.4]
    port_config=analog
    protocol_file=analog_loopstart_us.lec
    num_rings=2
    input_gain=0
    output_gain=0
    caller_id=enabled
    country=us600.qslac
[module.0/port.5]
    port_config=analog
    protocol_file=analog_loopstart_us.lec
    num_rings=2
    input_gain=0
    output_gain=0
    caller_id=enabled
    country=us600.qslac
[module.0/port.6]
    port_config=analog
    protocol_file=analog_loopstart_us.lec
    num_rings=2
    input_gain=0
    output_gain=0
    caller_id=enabled
    country=us600.qslac
[module.0/port.7]
    port_config=analog
    protocol_file=analog_loopstart_us.lec
    num_rings=2
    input_gain=0
    output_gain=0
    caller_id=enabled
    country=us600.qslac
[module.0/port.8]
    port_config=analog
    protocol_file=analog_loopstart_us.lec
    num_rings=2
    input_gain=0
    output_gain=0
    caller_id=enabled
    country=us600.qslac
```

BRI Port-Specific Configuration File Example

The following code shows an example configuration file for ports using ISDN BRI. The code specifies values for module number zero that automatically configure all the modules in the system to the settings in the file.

```
l3l4_trace=basic
l4l3_trace=basic
api_trace=verbose
internal_trace=basic
trace_file=call_control_trace.txt
max_trace_file_size=10

[module.0]
    pcm_law=alaw
[module.0/port.1]
    emulation=CPE
    port_config=BRI
    default_caller_id_channel_0=12344
    default_caller_id_channel_1=67890
    max_did_digits=0
    did_timeout=10
    reject_incomplete_did=false
[module.0/port.2]
    emulation=CPE
    port_config=BRI
    default_caller_id_channel_0=12344
    default_caller_id_channel_1=67890
    max_did_digits=0
    did_timeout=10
    reject_incomplete_did=false
```

E1 ISDN Port-Specific Configuration File Example

The following code shows an example configuration file for ports using E1 ISDN. The code specifies values for module number zero that automatically configure all the modules in the system to the settings in the file. This example also configures the trunks as fractional E1 lines, and sets each trunk with 15 channels available.

```
l3l4_trace=basic
l4l3_trace=basic
api_trace=verbose
internal_trace=basic
trace_file=call_control_trace.txt
max_trace_file_size=10

[module.0]
    pcm_law=alaw
    auto_connect=true
[module.0/clock_config]
    clock_mode=master
    clock_source=TrunkA
[module.0/port.1]
    emulation=CPE
    port_config=E1_ISDN
    line_coding=HDB3
    crc=enabled
    protocol=EURO
    max_did_digits=5
    did_timeout=10
    reject_incomplete_did=false
    default_caller_id=12344
    fractional_channel_count=15
    fractional_start_channel=0
[module.0/port.2]
    emulation=CPE
    port_config=E1_ISDN
    line_coding=HDB3
    crc=enabled
    protocol=EURO
    max_did_digits=5
    did_timeout=10
    reject_incomplete_did=false
    default_caller_id=12344
    fractional_channel_count=15
    fractional_start_channel=0
```

E1 R2 CAS Port-Specific Configuration File Example

The following code shows an example configuration file for ports using an R2 variant of the E1 CAS protocol. The code specifies values for module number zero that automatically configure all the modules in the system to the settings in the file.

```
l314_trace=none
l413_trace=none
api_trace=none
internal_trace=none
# Most of the time a path should be used for this file
name.
trace_file=ecc.log

[module.0]
    auto_connect=true
    pcm_law=alaw
[module.0/clock_config]
    clock_mode=master
    clock_source=TrunkA
[module.0/port.1]
    port_config=e1_r2_cas
# Most of the time a path should be used for this file
name.
    protocol_file=itu_china.r2
    CRC=enabled
    line_coding=hdb3
    line_impedance=120
    max_did_digits=4
[module.0/port.2]
    port_config=e1_r2_cas
# Most of the time a path should be used for this file
name.
    protocol_file=itu_china.r2
    CRC=enabled
    line_coding=hdb3
    line_impedance=120
    max_did_digits=4
```

T1 ISDN Port-Specific Configuration File Example

The following code shows an example configuration file for ports using T1 ISDN. The code specifies values for module number zero that automatically configure all the modules in the system to the settings in the file.

```
l3l4_trace=basic
l4l3_trace=basic
api_trace=verbose
internal_trace=basic
trace_file=call_control_trace.txt
max_trace_file_size=10

[module.0]
    pcm_law=mulaw
    auto_connect=true
[module.0/clock_config]
    clock_mode=master
    clock_source=TrunkA
[module.0/port.1]
    emulation=CPE
    port_config=T1_ISDN
    line_coding=B8ZS
    protocol=ATT
    max_did_digits=5
    did_timeout=10
    reject_incomplete_did=false
    default_caller_id=12344
[module.0/port.2]
    emulation=CPE
    port_config=T1_ISDN
    line_coding=B8ZS
    protocol=ATT
    max_did_digits=5
    did_timeout=10
    reject_incomplete_did=false
    default_caller_id=12344
```

T1 QSIG Port-Specific Configuration File Example

The following code shows an example configuration file for ports using T1 QSIG. The code specifies values for module number zero that automatically configure all the modules in the system to the settings in the file.

```
l3l4_trace=basic
l4l3_trace=basic
api_trace=verbose
internal_trace=basic
trace_file=call_control_trace.txt
max_trace_file_size=10

[module.0]
  channels=46
  set_api=bfv
  auto_connect=true
  pcm_law=mulaw
[module.0/clock_config]
  clock_mode=master
  clock_source=trunka
  clock_compatibility=none
  bus_speed=2
  master_ref_fallback=disabled
  master_drive=clock_a
[module.0/port.1]
  port_config=t1_qsig
  collision_priority=B
  call_diversion_completion_timer=3000
  default_caller_id=
  did_offset=0
  disable_alerting=true
  disable_call_proceed=false
  disable_conn_ack=false
  enable_call_diversion=false
  request_aoc=false
  fractional_channel_count=-1
  fractional_start_channel=0
  did_timeout=5
  max_overlapped_digits=20
  numbering_plan=unknown
  numbering_type=unknown
  call_type=auto
  presentation=allowed
```

```
  qsig_support=new
  sabme=true
  screening=user_not_screened
  send_dialcomplete=true
  transfer_variant=qsig
  wait_for_service_timeout=10
  emulation=slave
  line_coding=b8zs
  line_build_out=0_133
  max_did_digits=0
  reject_incomplete_did=false
[module.0/port.2]
  port_config=t1_qsig
  collision_priority=B
  call_diversion_completion_timer=3000
  default_caller_id=
  did_offset=0
  disable_alerting=true
  disable_call_proceed=false
  disable_conn_ack=false
  enable_call_diversion=false
  request_aoc=false
  fractional_channel_count=-1
  fractional_start_channel=0
  did_timeout=5
  max_overlapped_digits=20
  numbering_plan=unknown
  numbering_type=unknown
  call_type=auto
  presentation=allowed
  qsig_support=new
  sabme=true
  screening=user_not_screened
  send_dialcomplete=true
  transfer_variant=qsig
  wait_for_service_timeout=10
  emulation=slave
  line_coding=b8zs
  line_build_out=0_133
  max_did_digits=0
  reject_incomplete_did=false
```

T1 Robbed Bit Signaling Port-Specific Configuration File Example

The following code shows an example configuration file for ports using T1 with robbed bit signaling. The code specifies values for module number zero that automatically configure all the modules in the system to the settings in the file.

```
l3l4_trace=basic
l4l3_trace=basic
api_trace=verbose
internal_trace=basic
trace_file=call_control_trace.txt
max_trace_file_size=10

[module.0]
    pcm_law=mulaw
    auto_connect=true
[module.0/clock_config]
    clock_mode=master
    clock_source=TrunkA
[module.0/port.1]
    port_config=T1_ROBBED_BIT
    protocol_file=winkstart.lec
    line_coding=B8ZS
    line_type=ESF
    max_did_digits=4
    did_timeout=10
    reject_incomplete_did=false
[module.0/port.2]
    port_config=T1_ROBBED_BIT
    protocol_file=winkstart.lec
    line_coding=B8ZS
    line_type=ESF
    max_did_digits=4
    did_timeout=10
    reject_incomplete_did=
```

Examples of IP Call Control Configuration File

This section contains several coded examples of call control configuration files set up to support use of an IP call control stack:

Note: Only the TR1034 and SR140 support use of an IP call control stack.

- Single Module, Single SIP Stack — see *page 1308*
- Single Module, Single H.323 Stack — see *page 1310*
- Multiple Modules, Single Stack — see *page 1312*
- Multiple Modules, Multiple Stacks — see *page 1314*
- Single Virtual Module, Single Stack — see *page 1317*

Note: If a parameter specifies a file name, specify the full path to the file.
For example, if the install location is C:Brooktrout/Boston then specify the following parameter as:

```
protocol_file=C:/Brooktrout/Boston/config/
analog_loopstart_us.lec
```

Refer to *Sample Configuration Files on page 1318* for a listing of sample configuration files included with the Brooktrout SDK that provide a good starting point in your configuration. You might need to customize them for your system.

Single Module, Single SIP Stack

```
[host_module.1]
    module_library=brktsip.dll
    enabled=true
[host_module.1/t38parameters]
    fax_transport_protocol=t38_only
    t38_fax_rate_management=transferredTCF
    t38_max_bit_rate=33600
    t38_fax_version=3
    t38_fax_udp_ec=t38UDPRedundancy
    rtp_ced_enable=true
    media_renegotiate_delay_inbound=4000
    media_renegotiate_delay_outbound=-1
    t38_fax_fill_bit_removal=false
    t38_fax_transcoding_jbig=false
    t38_fax_transcoding_mmr=false
    t38_t30_fastnotify=true
    t38_UDPTL_redundancy_depth_control=5
    t38_UDPTL_redundancy_depth_image=2
[host_module.1/parameters]
    sip_Contact=0.0.0.0:0
    sip_description_URI=http:www.brooktrout.com
    sip_default_gateway=0.0.0.0:0
    sip_email=default@brooktrout.com
    sip_From=from@brooktrout.com
    sip_Max-Forwards=20
    sip_max_sessions=30
    sip_phone=+1-4085551212
    sip_proxy_server1=
    sip_proxy_server2=
    sip_proxy_server3=
    sip_registration_interval=60
    sip_registration_server1=
    sip_registration_server2=
    sip_registration_server3=
    sip_session_description=description_brooktrout
    sip_session_name=session_brooktrout
    sip_username=brooktrout

[host_module.1/rtp]
   rtp_frame_duration=20
   rtp_jitter_buffer_depth=100
   rtp_silence_control=inband
   rtp_type_of_service=0
   rtp_voice_frame_replacement=0
   rtp_codec=pcmu
   rtp_codec=pcma
```

```
[module.2]
[module.2/clock_config]
    clock_source=internal
    clock_mode=master
    clock_compatibility=none

[module.2/ethernet.1]
    dhcp=disabled
    ip_address=192.168.0.100
    ip_netmask=255.255.255.0
    ip_gateway=192.168.0.1
    ip_broadcast=192.168.0.255
    ip_arp_broadcast=10
    media_port_min=1000
    media_port_max=2000
    ethernet_speed=auto
[module.2/host_cc.1]
    host_module=1
    number_of_channels=24
```

Single Module, Single H.323 Stack

```
[host_module.1]
    module_library=brkth323.dll
    enabled=true
[host_module.1/t38parameters]
    t38_fax_rate_management=transferredTCF
    t38_max_bit_rate=14400
    t38_fax_udp_ec=t38UDPRedundancy
    rtp_ced_enable=true
    media_renegotiate_delay_inbound=4000
    media_renegotiate_delay_outbound=-1
    t38_fax_fill_bit_removal=false
    t38_fax_transcoding_jbig=false
    t38_fax_transcoding_mmr=false
    t38_t30_fastnotify=true
    t38_UDPTL_redundancy_depth_control=5
    t38_UDPTL_redundancy_depth_image=2
[host_module.1/parameters]
    h323_e164alias=5551212
    h323_e164alias=5553434
    h323_e164alias=5556767
    h323_default_gateway=208.242.16.158:1720
    h323_gatekeeper_id=
    h323_gatekeeper_ip_address=0.0.0.0:0
    h323_gatekeeper_ttl=10
    h323_h323IDalias=yourname
    h323_h323IDalias=companyname
    h323_local_ip_address=0.0.0.0:0
    h323_max_sessions=256
    h323_register=0
    h323_support_alternate_gk=0

[host_module.1/rtp]
   rtp_frame_duration=20
   rtp_jitter_buffer_depth=100
   rtp_silence_control=inband
   rtp_type_of_service=0
   rtp_voice_frame_replacement=0
   rtp_codec=pcmu
   rtp_codec=pcma

[module.2]
[module.2/clock_config]
    clock_source=internal
    clock_mode=master
    clock_compatibility=none
```

```
[module.2/ethernet.1]
    ip_address=192.168.0.100
    ip_netmask=255.255.255.0
    ip_gateway=192.168.0.1
    ip_broadcast=192.168.0.255
    ip_arp_timeout=600
    media_port_min=56000
    media_port_max=57000
    ethernet_speed=auto
[module.2/host_cc.1]
    host_module=1
    number_of_channels=24
```

Multiple Modules, Single Stack

```
[host_module.1]
    module_library=brktsip.dll
    enabled=true
[host_module.1/t38parameters]
    t38_fax_rate_management=transferredTCF
    t38_max_bit_rate=14400
    t38_fax_udp_ec=t38UDPRedundancy
    rtp_ced_enable=true
    media_renegotiate_delay_inbound=4000
    media_renegotiate_delay_outbound=-1
    t38_fax_fill_bit_removal=false
    t38_fax_transcoding_jbig=false
    t38_fax_transcoding_mmr=false
    t38_t30_fastnotify=true
    t38_UDPTL_redundancy_depth_control=5
    t38_UDPTL_redundancy_depth_image=2

[host_module.1/rtp]
   rtp_frame_duration=20
   rtp_jitter_buffer_depth=100
   rtp_silence_control=inband
   rtp_type_of_service=0
   rtp_voice_frame_replacement=0
   rtp_codec=pcmu
   rtp_codec=pcma

[host_module.1/parameters]
    sip_Contact=0.0.0.0:0
    sip_description_URI=http:www.brooktrout.com
    sip_default_gateway=0.0.0.0:0
    sip_email=default@brooktrout.com
    sip_From=from@brooktrout.com
    sip_Max-Forwards=20
    sip_max_sessions=30
    sip_phone=+1-4085551212
    sip_proxy_server1=
    sip_proxy_server2=
    sip_proxy_server3=
    sip_registration_interval=60
    sip_registration_server1=
    sip_registration_server2=
    sip_registration_server3=
    sip_session_description=description_brooktrout
    sip_session_name=session_brooktrout
    sip_username=brooktrout
```

```
[module.2]
[module.2/clock_config]
    clock_source=internal
    clock_mode=master
    clock_compatibility=none
[module.2/ethernet.1]
    ip_address=192.168.0.100
    ip_netmask=255.255.255.0
    ip_gateway=192.168.0.1
    ip_broadcast=192.168.0.255
    ip_arp_timeout=600
    media_port_min=56000
    media_port_max=57000
    ethernet_speed=auto
[module.2/host_cc.1]
    host_module=1
    number_of_channels=24

[module.3]
[module.3/clock_config]
    clock_source=internal
    clock_mode=master
    clock_compatibility=none
[module.3/ethernet.1]
    ip_address=192.168.0.100
    ip_netmask=255.255.255.0
    ip_gateway=192.168.0.1
    ip_broadcast=192.168.0.255
    ip_arp_timeout=600
    media_port_min=56000
    media_port_max=57000
    ethernet_speed=auto
[module.3/host_cc.1]
    host_module=1
    number_of_channels=24
```

Multiple Modules, Multiple Stacks

```
[host_module.1]
    module_library=brktsip.dll
    enabled=true

[host_module.1/t38parameters]
    t38_fax_rate_management=transferredTCF
    t38_max_bit_rate=14400
    t38_fax_udp_ec=t38UDPRedundancy
    rtp_ced_enable=true
    media_renegotiate_delay_inbound=4000
    media_renegotiate_delay_outbound=-1
    t38_fax_fill_bit_removal=false
    t38_fax_transcoding_jbig=false
    t38_fax_transcoding_mmr=false
    t38_t30_fastnotify=true
    t38_UDPTL_redundancy_depth_control=5
    t38_UDPTL_redundancy_depth_image=2

[host_module.1/rtp]
   rtp_frame_duration=20
   rtp_jitter_buffer_depth=100
   rtp_silence_control=inband
   rtp_type_of_service=0
   rtp_voice_frame_replacement=0
   rtp_codec=pcmu
   rtp_codec=pcma
[host_module.1/parameters]
    sip_Contact=0.0.0.0:0
    sip_description_URI=http:www.brooktrout.com
    sip_default_gateway=0.0.0.0:0
    sip_email=default@brooktrout.com
    sip_From=from@brooktrout.com
    sip_Max-Forwards=20
    sip_max_sessions=30
    sip_phone=+1-4085551212
    sip_proxy_server1=
    sip_proxy_server2=
    sip_proxy_server3=
    sip_registration_interval=60
    sip_registration_server1=
    sip_registration_server2=
    sip_registration_server3=
    sip_session_description=description_brooktrout
    sip_session_name=session_brooktrout
    sip_username=brooktrout
```

```
[host_module.2]
    module_library=brkth323.dll
    enabled=true
[host_module.2/t38parameters]
    t38_fax_rate_management=transferredTCF
    t38_max_bit_rate=14400
    t38_fax_udp_ec=t38UDPRedundancy
    rtp_ced_enable=true
    media_renegotiate_delay_inbound=4000
    media_renegotiate_delay_outbound=-1
    t38_fax_fill_bit_removal=false
    t38_fax_transcoding_jbig=false
    t38_fax_transcoding_mmr=false
    t38_t30_fastnotify=true
    t38_UDPTL_redundancy_depth_control=5
    t38_UDPTL_redundancy_depth_image=2

[host_module.2/rtp]
   rtp_frame_duration=20
   rtp_jitter_buffer_depth=100
   rtp_silence_control=inband
   rtp_type_of_service=0
   rtp_voice_frame_replacement=0
   rtp_codec=pcmu
   rtp_codec=pcma

[host_module.2/parameters]
    h323_e164alias=5551212
    h323_e164alias=5553434
    h323_e164alias=5556767
    h323_default_gateway=0.0.0.0:0
    h323_gatekeeper_id=
    h323_gatekeeper_ip_address=0.0.0.0:0
    h323_gatekeeper_ttl=10
    h323_h323IDalias=yourname
    h323_h323IDalias=companyname
    h323_local_ip_address=0.0.0.0:0
    h323_max_sessions=256
    h323_register=0
    h323_support_alternate_gk=0

[module.2]
[module.2/clock_config]
    clock_source=internal
    clock_mode=master
    clock_compatibility=none
[module.2/ethernet.1]
    dhcp=disabled
```

```
    ip_address=192.168.0.100
    ip_netmask=255.255.255.0
    ip_gateway=192.168.0.1
    ip_broadcast=192.168.0.255
    ip_arp_broadcast=10
    media_port_min=1000
    media_port_max=2000
    ethernet_speed=auto
[module.2/host_cc.1]
    host_module=1
    number_of_channels=24

[module.3]
[module.3/clock_config]
    clock_source=internal
    clock_mode=master
    clock_compatibility=none
[module.3/ethernet.1]
    dhcp=disabled
    ip_address=192.168.0.100
    ip_netmask=255.255.255.0
    ip_gateway=192.168.0.1
    ip_broadcast=192.168.0.255
    ip_arp_broadcast=10
    media_port_min=1000
    media_port_max=2000
    ethernet_speed=auto
[module.3/host_cc.2]
    host_module=2
    number_of_channels=24
```

Single Virtual Module, Single Stack

```
[module.41]
# This parameter should be modified to point to the correct location of the
# bostvb.dll
    vb_firm=C:\Brooktrout\Boston\fw\bostvb.dll
# This parameter should be set to the number of channels licensed for the
# SR140 product
    channels=4


[module.41/ethernet.1]
       ip_interface=
       media_port_min=56000
       media_port_max=57000
[module.41/host_cc.1]
       host_module=1
       number_of_channels=4
[host_module.1]
    module_library=brktsip.dll
    enabled=true
[host_module.1/t38parameters]
    t38_fax_rate_management=transferredTCF
    t38_max_bit_rate=14400
    t38_fax_udp_ec=t38UDPRedundancy
    rtp_ced_enable=true
    media_renegotiate_delay_inbound=1000
    media_renegotiate_delay_outbound=2000
    t38_fax_fill_bit_removal=false
    t38_fax_transcoding_jbig=false
    t38_fax_transcoding_mmr=false
    t38_t30_fastnotify=true
    t38_UDPTL_redundancy_depth_control=5
    t38_UDPTL_redundancy_depth_image=2

[host_module.1/rtp]
   rtp_frame_duration=20
   rtp_jitter_buffer_depth=100
   rtp_silence_control=inband
   rtp_type_of_service=0
   rtp_voice_frame_replacement=0
   rtp_codec=pcmu
   rtp_codec=pcma

[host_module.1/parameters]
    sip_Contact=0.0.0.0:0
    sip_description_URI=http:www.brooktrout.com
```

```
sip_default_gateway=0.0.0.0:0
sip_email=default@brooktrout.com
sip_From=from@brooktrout.com
sip_Max-Forwards=20
sip_max_sessions=30
sip_phone=+1-4085551212
sip_proxy_server1=
sip_proxy_server2=
sip_proxy_server3=
sip_registration_interval=60
sip_registration_server1=
sip_registration_server2=
sip_registration_server3=
sip_session_description=description_brooktrout
sip_session_name=session_brooktrout
sip_username=brooktrout
```

Sample Configuration Files

The following are sample configuration files (with brief descriptions) that provide a good starting point for your configuration. You might need to customize them for your system.

Several call control configuration files, for example "callctrl.cfg", are provided in the config directory for some of the most common configurations.

The .lec, .qslac, and .r2 files may also be required for use for call control configuration. They are referred to by name from within the call control configuration files. For a quick start, copy all files to a test directory and then choose the files to configure.

callctrl.cfg

This is an all-in-one file that contains examples for several different types of boards. All of the configuration lines have been commented out. You should uncomment the lines that are appropriate for your configuration.

callctrl_analog.cfg

8 loop-start analog ports

Each port connected to a logical channel

callctrl_bri.cfg

BRI

All BRI channels are connected through to the logical channels.

callctrl_did.cfg

DID

4 winkstart analog DID ports

All DID channels are connected through to the logical channels.

callctrl_e1.cfg

E1

Robbed Bit Signaling

H.100 bus master

HDB3

120 Ohms

All E1 timeslots are connected through to the logical channels.

callctrl_e1_isdn.cfg

E1

ISDN

Bus master

HDB3

All E1 timeslots are connected through to the logical channels.

callctrl_e1_qsig.cfg

E1

QSIG

Bus master

HDB3

All QSIG timeslots are connected through to the logical channels.

callctrl_h323.cfg

Board based with host based H.323 stack

t.38

callctrl_h323_hbf.cfg

Host based H.323

t.38

callctrl_r2.cfg

R2 CAS

HDB3

CRC

120-Ohm

Bus master

All E1 timeslots are connected through to the logical channels.

callctrl_sip.cfg

Board based with host based SIP stack

t.38

callctrl_sip_hbf.cfg

Host based SIP

t.38

callctrl_t1.cfg

T1

Robbed Bit Signalling

E&M wink start

Bus master

Extended Superframe (ESF)

B8ZS

All T1 timeslots are connected through to the logical channels.

callctrl_t1_isdn.cfg

T1

ISDN

Bus master

Extended Superframe (ESF)

B8ZS

All T1 timeslots are connected through to the logical channels.

callctrl_t1_qsig.cfg

T1

QSIG

Bus master

HDB3

All QSIG timeslots are connected through to the logical channels.

Routing Table Configuration File

The optional routing table configuration file is an ASCII file that contains one or more routing rules which define how inbound calls should be routed. The ***routing_table*** parameter in the *Call Control Configuration File* (see *page 1170*) specifies the path and filename of the routing table configuration file.

This section describes the content of the routing table configuration file as follows:

- ***Routing Table Configuration File Format***
- ***Routing Rule Parameters***
- ***Examples of Routing Table Configuration Files***

Note: The Routing Table Configuration file only supports routing rules for inbound calls using the SIP IP call control protocol and can only route these calls to channels on SR140 modules.

Routing Table Configuration File Format

The general format of the file is:

Routing Rule Parameters

Parameters that specify routing rule settings.

```
[routing.#]
   channel=XXXX
   called_address=XXXX
   calling_address=XXXX
```

All routing rule indexes and channel numbers are specified as decimal numbers. Text strings are not case sensitive and only use ASCII format. Comment lines in the file should start with a ‘;’ (semicolon) or a ‘#’ (pound or number) symbol.

routing.#

The *routing* keyword allows users to specify the start of a new routing rule in the routing table configuration file.

Each routing rule has a 1-based index associated with it. Routing rules also have a priority associated with them with `[routing_rule.1]` being the highest priority, `[routing_rule.2]` being the second highest priority, etc.

When an inbound SIP call is received and inbound call routing is enabled, an attempt will be made to match the call to one of the configured routing rules starting with the highest priority routing rule and ending with the lowest priority routing rule. If a match is found, the call will be routed to the SR140 channel that's been waiting the longest for an inbound call and is assigned to the routing rule.

Within the routing table configuration file, the routing rules do not need to be specified in consecutive order.

For example, the routing rules could be specified in a configuration file in the following order: 1, 2, 99:

```
[routing_rule.1]
.
.
.
[routing_rule.2]
.
.
.
[routing_rule.99]
.
.
.
```

A maximum of 100 routing rules may be specified in a routing table configuration file.

Routing Rule Parameters

Set the following parameters to define configuration information that applies to the routing rule (*routing.#*).

Parameter	Value

channel	The *channel* parameter is required for all valid routing rules and allows users to specify an SR140 channel or range of SR140 channels that a routing rule applies to. When a single channel is specified, the format for the channel parameter is:
	`channel=<number>`
	When a range of channels is specified, the format is:
	`channel=<min> - <max>`
	A routing rule can have *channel* parameter values that overlap channels specified for a different routing rule. For example, `[routing_rule.1]` could specify channels 0 through 4 and `[routing_rule.2]` could specify channels 3 through 4.
	If a routing rule is encountered that doesn't have a *channel* parameter specified, the routing rule will be considered invalid and won't be used at runtime to route inbound calls. The Boston Host Service will not treat this as a severe error and will successfully initialize when one or more invalid routing rules are detected.
	If, during initialization, all of the specified routing rules are determined to be invalid, then the Boston Host Service will initialize successfully with inbound call routing functionality disabled.
	Range: 0 through 512
	Default: N/A
	Note: While the maximum supported channel number for routing rules is 512, the maximum valid channel number for any specific configuration is the highest ordinal number for an SR140 channel. For example, if a system has two SR140 modules in a system with each module configured for 4 channels, then the valid range of channel parameter values would be 0 to 7.

Parameter	Value
called_address	The *called_address* parameter allows users to specify a string containing a pattern to use when matching inbound calls to routing rules. The called address of an inbound SIP call is extracted from the *To* SIP header. For example, if an inbound SIP call contains the following *To* header:

```
To: <sip:1001@10.128.22.6:5060>
```

The called address of the call would be “1001” and this is what the pattern specified by the *called_address* parameter of a routing rule would be matched against.

The pattern specified for the *called_address* parameter value can consist of any of the decimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) as well as the special characters identified in the following table:

Dial Plan Character	*Character Description*
x or X	Matches any digit from 0-9
[	Specify start of digit range
-	Specify range of digits
]	Specify end of digit range

Match ranges are created using [min-max]. A left bracket, hyphen (-) and a right bracket are required. Minimum and maximum values may range from 0 through 9. Brackets may be repeated, but not nested.

An empty *called_address* string (or a routing rule where the *called_address* parameter isn't specified) indicates that the *called_address* information associated with the inbound call should not be checked against this parameter setting. If the inbound call matches the *calling_address* pattern of a routing rule and the *called_address* parameter value for the same routing rule is an empty string, this would constitute a match and the call would be routed to one of the SR140 channels associated with this routing rule.

A routing rule that has both the *called_address* and *calling_address* parameters set to empty strings is invalid and will not be used at runtime for routing inbound calls.

Range: 1-255 characters

Value Type: character string (up to 256 characters)

Default: <blank> (empty string)

Parameter	Value
calling_address	The *calling_address* parameter allows users to specify a string containing a pattern to use when matching inbound calls to routing rules. The calling address of an inbound SIP call is extracted from the *From* SIP header. For example, if an inbound SIP call contains the following *From* header:

```
From: <sip:7814499009@10.128.22.6:5060>;tag=XXXXXX
```

The calling address of the call would be “7814499009” and this is what the pattern specified by the *calling_address* parameter of a routing rule would be matched against.

The pattern specified for the *calling_address* parameter value can consist of any of the decimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) as well as the special characters identified in the following table:

Dial Plan Character	*Character Description*
x or X	Matches any digit from 0-9
[	Specify start of digit range
-	Specify range of digits
]	Specify end of digit range

Match ranges are created using [min-max]. A left bracket, hyphen (-) and a right bracket are required. Minimum and maximum values may range from 0 through 9. Brackets may be repeated, but not nested.

An empty *calling_address* string (or a routing rule where the *calling_address* parameter isn't specified) indicates that the *calling_address* information associated with the inbound call should not be checked against this parameter setting. If the inbound call matches the *called_address* pattern of a routing rule and the *calling_address* parameter value for the same routing rule is an empty string, this would constitute a match and the call would be routed to one of the SR140 channels associated with this routing rule.

A routing rule that has both the *called_address* and *calling_address* parameters set to empty strings is invalid and will not be used at runtime for routing inbound calls.

Range: 1-255 characters

Value Type: character string (up to 256 characters)

Default: <blank> (empty string)

Examples of Routing Table Configuration Files

This section contains an example of a routing table configuration file set up to specify inbound call routing rules:

- ***Routing Rules for Eight Channel Configuration***

Routing Rules for Eight Channel Configuration

The following routing table configuration file example contains valid routing rules for a system that has 8 SR140 channels configured:

```
# Rule to match inbound calls that have a called
# address of "1001" and calling address that's
# 10 digits long and starts with "781449".
# Route matching calls to SR140 channel 0
[routing.1]
   channel 0
   called_address = "1001"
   calling_address= "781449XxXx"
# Rule to match inbound calls that have a called
# address that's 4 digits long and begins with '2'
# and ends with '5'. Ignore the calling address
# when checking if the inbound call matches this
# routing rule. Route matching calls to SR140
# channels 0 or 1.
[routing.2]
   channel 0-1
   called_address = "2xx5"
   calling_address =""
# Rule to match inbound calls that have a called
# address that's 4 digits long and begins with
# either '2' or '3' and ends with "234" (i.e.
# "2234" or "3234"). The calling address must also
# be 10 digits long and start with "973". Route
# matching calls to SR140 channels 2, 3, 4, or 5.
[routing.3]
   channel 2-5
   called_address = "[2-3]234"
   calling_address="973xxxXXXX"
# Rule to match inbound calls that have a called
# address that's 4 digits long and begins with
# '4'. The second digit must be '0', '1', '2',
# '3' or '4'. The third digit must be '8' or
# '9' and the fourth digit must be '5', '6', '7',
# '8' or '9'. The calling address must also be 11
# digits long and set to "18007554444". Route
# matching calls to SR140 channels 0 thru 7.
[routing.4]
   channel 0-7
   called_address="4[0-4][8-9][5-9]"
   calling_address="18007554444"
```

```
# Rule to match inbound calls that have a called
# address that's 7 digits long. Since the
# calling_address parameter isn't specified for
# this routing rule, ignore the calling address
# when checking if the inbound call matches this
# routing rule. Route matching calls to SR140
# channel 4. Note that routing rule indexes do
# not have to be consecutive.
[routing.50]
   channel 4
   called_address="XXXXXXX"
# Rule to match inbound calls that have a calling
# address that's 11 digits long and begins with
# "1781". Since the called_address parameter
# isn't specified for this routing rule, ignore
# the called address when checking if the inbound
# call matches this routing rule. Route matching
# calls to SR140 channels 3 thru 7.
[routing.60]
   channel 3-7
   calling_address="1781xxxxxxx"
# Rule to match inbound calls that have a calling
# address that's 7 digits long and begins with
# '4' followed by '7', '8' or '9'followed by '1'
# or '2' followed by 4 other digits. (e.g. 4712468
# or 4821357). Since the called_address parameter
# is set to an empty string, ignore the called
# address when checking if the inbound call
# matches this routing rule. Route matching calls
# to SR140 channel 6.
[routing.100]
   channel 6
   called_address=""
   calling_address="4[7-9][1-2]XXXX"
```

SRTP Configuration File

The ***srtp_config_filename*** in the *Call Control Configuration File* (see *page 1170*) specifies the path and filename of the SRTP configuration file.

This section describes the content of the SRTP configuration file. This file is an ASCII file that contains key parameters and values used to configure SRTP.

Key Name	Description
srtp_accept	With SRTP enabled, the SR140 can be configured to reject SRTP by setting this parameter to FALSE. The following are the allowable parameter values: TRUE Both RTP and SRTP streams accepted. FALSE RTP only accepted. *Default:* TRUE
srtp_enforce	With SRTP enabled, the SR140 can be configured to reject RTP by setting this parameter to TRUE. The following are the allowable parameter values: TRUE SRTP only accepted. FALSE Both RTP and SRTP streams accepted. *Default:* TRUE
srtp_crypto_suite	The crypto-suite field is an identifier that describes the encryption and authentication algorithm used for SRTP. *Default:* AES_CM_128_HMAC_SHA1_80
srtp_master_key_len	The length (in bits) of the master key. The only value currently supported is 128 for use with AES_CM_128_HMAC_SHA1. *Default:* 128
srtp_salting_key_len	The length in bits of the master salt key. The only value currently support is 112. *Default:* 112
srtp_num_keys	The number of master keys generated and associated with each outgoing SRTP stream. *Default:* 1

Key Name	Description
srtp_mki_len	The length in bytes of the Master Key Identifier (MKI) associated with the SRTP master key. *Default:* 4
srtp_lifetime	The lifetime of the master keys, that is, the maximum number of SRTP packets that need to be secured with the master key. Default is 48, corresponding to a lifetime of 2^48. *Default:* 48
srtcp_lifetime	The lifetime of the master keys, that is, the maximum number of SRTCP packets that need to be secured with the master key. Default is 31, corresponding to a lifetime of 2^31. *Default:* 31
srtp_kdr	The Key Derivation Rate (KDR) is the rate at which session keys are derived from the master key and master salt key. If KDR is set to 0, the key is derived only once. This field takes an integer value in the range 1 to 24, which corresponds to a KDR value in the range 2^1 to 2^24. The following are the allowable parameter values: TRUE Both RTP and SRTP streams accepted. FALSE RTP only accepted. *Range:* 0 (No MKI), 1 - 24 *Default:* 0
srtp_window_size	A parameter that protects against replay attacks (that is, the capturing of a packet and later reinsertion into a stream). *Range:* 64 - 256 *Default:* 64
srtp_unencrypted_flag	Enables or disables the encryption of SRTP packets or the use of the NULL cipher in SRTP. The following are the allowable parameter values: TRUE SRTP messages are not encrypted. FALSE SRTP are encrypted. *Default:* FALSE
srtcp_unencrypted_flag	Enables or disables the encryption of SRTCP packets or the use of the NULL cipher in SRTCP. The following are the allowable parameter values: TRUE SRTCP messages are not encrypted. FALSE SRTCP are encrypted. *Default:* FALSE

Key Name	Description
srtp_unauthenticated_flag	Enables or disables SRTP authentication. The following are the allowable parameter values: TRUE SRTP messages are not authenticated. FALSE SRTP messages are authenticated. *Default:* FALSE

TLS Configuration File

The ***tls_config_filename*** in the *Call Control Configuration File* (see *page 1170*) specifies the path and filename of the TLS configuration file.

This section describes the content of the TLS configuration file.

Key Name	Description
sip_tls_method	TLS mode version. The following are the allowable parameter values: tls1.2 tls1.1 tls1.0 sslv3 *Default:* tls1.2
local_rsa_private_key_filename	Full path and filename of the RSA private key. FULLPATH Contains the full path to the TLS configuration file. *Default:* character string
local_rsa_private_key_password	Password for the RSA certificate. *Value Type:* character string *Default:* null
local_rsa_cert_filename	Full path and filename of the RSA certificate. FULLPATH Contains the full path to the TLS configuration file. *Value Type:* character string
local_dss_private_key_filename	Full path and filename of the DSS private key. FULLPATH Contains the full path to the TLS configuration file. *Value Type:* character string
local_dss_private_key_password	Password for the DSS certificate. *Value Type:* character string *Default:* null
local_dss_cert_filename	Full path and filename of the DSS certificate. FULLPATH Contains the full path to the TLS configuration file. *Value Type:* character string

Key Name	Description
ca_cert_number	Number of root CA certificates. *Range:* 0 - 128 *Default:* 0
ca_cert_filename	Full path and filename of the root CA certificate. FULLPATH Contains the full path to the TLS configuration file. *Value Type:* character string
chain_cert_number	Number of chained certificates. *Range:* 0 - 128 *Default:* 0
chain_cert_filename	Full path and filename of the chained certificate. FULLPATH Contains the full path to the TLS configuration file. *Value Type:* character string
crl_number	Number of certificate revocation list files. *Range:* 0 - 128 *Default:* 0
crl_filename	Full path and filename of the certificate revocation list. FULLPATH Contains the full path to the TLS configuration file. *Value Type:* character string
local_cipher_suite	Local cipher suite string. This string defines the allowed local cipher suites to be used by the SR140. See *Volume 6, Appendix I, SR140 Security Capabilities* for further details. *Default:* "ALL:!EXPORT:!LOW:!aNULL:!eNULL:!SSLv2"
dh_param_512_filename	Filename to override generation of new DH key. Filename points to a PEM format file that contains DH parameters with 512-bit key. *Default:* SR140 will use the pre-build default DH parameters for DH key exchanger ciphers.
dh_param_1024_filename	Filename to override generation of new DH key. Filename points to a PEM format file that contains DH parameters with 1024-bit key. *Default:* SR140 will use the pre-build default DH parameters for DH key exchanger ciphers.
dh_param_2048_filename	Filename to override generation of new DH key. Filename points to a PEM format file that contains DH parameters with 2048-bit key. *Default:* SR140 will use the pre-build default DH parameters for DH key exchanger ciphers.

Key Name	Description
session_id	An application may optionally enable server session caching by setting session_id string. If the string is set, the SR140 will enable session caching on server side and session identifier will be provided to client during handshake. During a new handshake, if session_id in ClientHello is non-empty, the SR140 will look up the session cache for a match and resume a session if possible. The session timeout is 300 seconds and currently not configurable. Server session cache terminates when the host module is terminated. Default value of session_id is NULL and server returns an empty session_id to indicate that the session will not be cached and therefore cannot be resumed. *Default:* NULL
client_cert_required	During TLS handshake, TLS server may optionally require client certificate for mutual authentication. If enabled and the client fails to present its certificate or certificate verification fails, the handshake will fail. The following are the allowable parameter values: TRUE Application requires client certificate. FALSE Application does not require client certificate. *Default:* FALSE
allow_self_signed_certs	Allow the usage of self-signed certificates. The following are the allowable parameter values: TRUE Allow self-signed certificates. FALSE Do not allow self-signed certificates. *Default:* FALSE

Parameters for Technical Support Purposes

The Bfv API provides the following parameters that might be used when getting help from Dialogic Technical Services and Support. These parameters should never be used unless explicitly directed to do so by Dialogic Technical Services and Support. If directed to use one of these parameters, enter the parameter and its value into the user-defined configuration file (see *page 1152*).

Parameter	Purpose
adpcm_exp	Directs how the firmware handles certain distortions in the data during playback of ADPCM (ADPCM expansion). `0` Bias removal. `1` High pass filter and bias removal. `2` "Leak" correction mode. *Range:* `0, 1, 2` *Value Type:* decimal *Default:* `1`
cp_silence_duration	Specifies the length of time in ms for a SILENCE call progress value to be generated when using CALL_PROTOCOL_VOICE or CALL_PROTOCOL_VOICE_NO_RAW. *Unit:* ms *Value Type:* decimal *Default:* 8000
debug_control	Turns on firmware debugging features. Supply two values – a facility number and a debug value. The *debug_control* and *debug_var_control* parameters may appear up to four times. *Value Type:* hexadecimal *Default:* none
debug_var_control	Sets firmware debugging variables. Supply three values - a facility number, a variable ID, and a debug value. The *debug_var_control and debug_control* parameters may appear up to four times. *Value Type:* hexadecimal *Default:* none
dtmf_rdelta	Defines allowed fluctuations of the DTMF level. *Range:* 0 – 10 *Value Type:* decimal *Default:* `0`

Parameter	Purpose
error_enable	Turns error detection on (1) or off (0) during fax reception in non-ECM mode. *Range:* 0 to 1 *Value Type:* decimal *Default:* 1 (enabled)
post_dialing_enable	Enables or disables the feature for post-dialing of DTMF digits. See *BfvLineOriginateCall args.phonenum* for a description of this feature. 0 Turns post-dialing off. 1 Turns post-dialing on. *Range:* 0 or 1 *Value Type:* decimal *Default:* 1
play_cdm	Stops dynamic range control (DRC), gain, and preemphasis together if set to 0. *Range:* 0 or 1 *Value Type:* decimal *Default:* 1
play_drc	Specifies whether dynamic range control (DRC) is on during speech playback. 0 DRC off. 1 DRC on. *Range:* 0 or 1 *Value Type:* decimal *Default:* 1
play_preemph	Specifies the mode for high frequency preemphasis during speech playback. 0 Preemphasis is off. 1 Preemphasis version 1 is used. 2 Preemphasis version 2 is used. The Bfv API performs pre-emphasis during playback. A pre-emphasis filter processes the outgoing signal, changing its frequency characteristics. The filter amplifies high frequency components of the outgoing signal, with the larger filter numbers indicating greater amplification. Preemphasis compensates for the loss of high frequencies that sometimes happens on analog lines. For T1, always select 0. *Range:* 0, 1, 2 *Value Type:* decimal *Default:* 0

Parameter	Purpose
rcv_level_override	Specifies the overriding of the REC_LEVEL_XXX level specified in the BT_CPARM.CFG file where XXX depends on the country code. The override will be relative, so this parameter will either add or subtract from the BT_CPARM.CFG level. This value is in 0.5 dB units. If this value is positive it will add to the REC_LEVEL_XXX level and if it is negative it will subtract from the REC_LEVEL_XXX level. For example, if this value is set to 3 the receive level will be increased by 1.5 dB and if this value is set to -3 the receive level will be decreased by 1.5 dB. *Value Type:* 0.5 dB *Default:* 0
T2_timer	Specifies the length of the T2 timer. The ITU-T (previously CCITT) T.30 specification contains the definition for the T2 timer. If this keyword is not in the user configuration file or has a value of 0 the T2 timer will use a default value of 6 seconds for PSTN and G.711 RTP and a default value of 6.95 seconds for T.38. If this keyword has a value from 3000 to 90000 ms the T2 timer will use this value for the T2 timeout. *Unit:* ms *Range:* 0 and 3000 - 90000 *Value Type:* decimal *Default:* 0
t4_recv_timer	Specifies the length of the T4 timer. It is used to set the T4 timer on the answer side if it is set to a non-zero value in the user configuration file. See the keyword *t4_timer_mode* for more information. The ITU-T (previously CCITT) T.30 specification contains the definition for the T4 timer. *Unit:* ms *Range:* 0 and 3000 - 15000 *Value Type:* decimal *Default:* 0
t4_xmit_timer	Specifies the length of the T4 timer. It is used to set the T4 timer on the originate side if it is set to a non-zero value in the user configuration file. See the keyword *t4_timer_mode* for more information. The ITU-T (previously CCITT) T.30 specification contains the definition for the T4 timer. *Unit:* ms *Range:* 0 and 3000 - 15000 *Value Type:* decimal *Default:* 0

Parameter	Purpose
t4_increment	Specifies the T4 timer increment value used for mode 1 (see *t4_timer_mode*) for the T4 timer initialization. *Unit:* ms *Range:* 100 - 1000 *Value Type:* decimal *Default:* 500
t4_timer_mode	Specifies the mode for the T4 timer. Mode 0 uses adaptation with a fixed 500 ms increment. If the value for *t4_recv_timer* on the answer side or *t4_xmit_timer* on the originate side is set to a non-zero value, adaptation is disabled. Mode 1 uses a programmable increment for the T4 timer initialization. The *t4_increment* value is used for initializing the second T4 timer depending on the value of *t4_recv_timer* on the answer side, *t4_xmit_timer* on the originate side, the media type, and automatic or manual fax operation. Adaptation is always disabled for this mode. Mode 2 allows the host application to adapt the T4 timer value via an API function. In this mode, the firmware will send an event (see the function *BfvFaxT4TimerParams(lp, args)*) to the host which informs the host application of the attempt number which just took place, the duration to receive the last response or the T4 timeout value if no response is received, the current value of the T4 timer setting and the T4 timeout flag indicating if a response was received or the T4 timer expired. The host application can then set the T4 timer value (limited by the firmware to 3,000 to 15,000 ms) with the *BfvFaxT4TimerParams* API function for the next command response exchange. *Unit:* unitless *Range:* 0 - 2 *Value Type:* decimal *Default:* 0
ud_reset_timer	Specifies a value that determines whether the DSP watchdog is enabled on digital TR1034 boards. 0 Off. 1 On. *Range:* 0 or 1 *Value Type:* decimal *Default:* 0

Parameter	Purpose
v34_transmit_only	Specifies a value that determines whether the application transmits and receives in V.34 mode or only transmits in V.34 mode (fax reception occurs in V.17 mode). To use this parameter, you must set the *v34_enabled* parameter (see *page 1168*) to a value of 1 (enabled) and enable the *V34Enab* feature key value. `0` Allows reception and transmission in V.34 mode. `1` Only allows transmission in V.34 mode. *Range:* 0 or 1 *Value Type:* decimal *Default:* `0`
xmt_level_override	Specifies the overriding of the DATA_LEVEL_XXX level specified in the BT_CPARM.CFG file where XXX depends on the country code. The override will be relative, so this parameter will either add or subtract from the BT_CPARM.CFG level. This value is in 0.5 dB units. If this value is positive it will add to the DATA_LEVEL_XXX level and if it is negative it will subtract from the DATA_LEVEL_XXX level. For example, if this value is set to 3 the transmit level will be increased by 1.5 dB and if this value is set to -3 the transmit level will be decreased by 1.5 dB. *Value Type:* 0.5 dB *Default:* `0`

B - Bfv API Structures

This appendix provides details about data structures used within the Bfv API.

It has the following sections:

- *Address Structure*
- *Result Structures*
- *DCS and DIS/DTC Info Structures*

Address Structure

Several functions and macros use the type `MILL_ADDR`, which is a structure representing a Millennium (TR1000 Series board) address. It contains the following fields:

```
typedef struct {
    unsigned char mm_bFacility;
    unsigned char mm_bChannel;
    unsigned char mm_bModule;
    unsigned char mm_bMachine;
} MILL_ADDR;
```

The facility, channel, module, and machine components of an address are represented by the similarly named components of the structure.

Result Structures

The following result structures and their return values are used by many functions:

```
typedef struct {
    int status;
    int line_status;
} RES;

typedef struct {
    int call_type;
    char dest_id[MAX_DID];
    /* The rest are ISDN only */
#define called_party_number dest_id
    char called_party_subaddress[MAX_DID];
    char calling_party_number[MAX_DID];
    char calling_party_subaddress[MAX_DID];
    char redir_number[MAX_DID];
    int redir_reason;
    char name_ident;
    int name_char_set;
    char connected_num[MAX_CONN_NUM];
} CALL_RES;

typedef struct {
    char remote_id[21];
    unsigned char nsf_nss_frame[MAX_NSF];
    char subaddress[21];
    char password[21];
    char selective_polling[21];
} INFO_RES;
```

```
typedef struct {
    int page_complete_type;
    int continue_breaks;
    unsigned ascii_bytes;
    unsigned bad_lines;
    unsigned total_lines;
    unsigned total_rcv;
    unsigned bit_rate;
    unsigned page_status;
    unsigned char pc_fifo[PC_FIFO_NBYTES];
    unsigned sig_level;
    unsigned line_noise;
    unsigned sig_quality;
    unsigned char lp_fifo[LP_FIFO_NBYTES];
    unsigned char confirm_value;
    int direction;
    int resolution;
    int width;
    unsigned long eff_page_type;
    struct page_res *next;
    unsigned bad_lines_NS;
    unsigned bad_lines_N23;
    unsigned bad_lines_N4;
    unsigned num_ECM_frames;
    unsigned num_PPR;
    unsigned num_PPR_NS;
    unsigned num_PPR_N23;
    unsigned num_PPR_N4;
    unsigned time_t4;
    unsigned time_training;
    unsigned iaf_initial_speed;
    unsigned iaf_final_speed;
} PAGE_RES;
typedef struct {
    int number_of_pages;
```

```
int bad_pages;
char remote_id[21];
long duration;
PAGE_RES *reslist_head;
PAGE_RES *reslist_tail;
unsigned initial_bit_rate;
unsigned ffom;
unsigned time_t1;
unsigned time_t38;
unsigned termination_phase;
unsigned count_RTP;
unsigned count_RTN;
unsigned count_CRP;
unsigned count_CTC;
enum TRxMediaType media_type;
unsigned RTPRcvrPackets;
unsigned RTPSenderPackets;
unsigned RTPRcvrLostPackets;
unsigned RTPRcvrLostPackets_NS;
unsigned RTPRcvrLostPackets_N23;
unsigned RTPRcvrLostPackets_N4;
unsigned RTPRcvrJitter;
unsigned RTPRcvrOOSPackets;
unsigned RTPRcvrRedundancyLevel;
unsigned T38RcvrPackets;
unsigned T38RcvrOctets;
unsigned T38SenderPackets;
unsigned T38SenderOctets;
unsigned T38RcvrLostPackets;
unsigned T38RcvrLostPackets_NS;
unsigned T38RcvrLostPackets_N23;
unsigned T38RcvrLostPackets_N4;
unsigned T38RcvrOOSPackets;
unsigned T38RcvrUsedRedundacy;
unsigned T38RcvrImageRedundancyLevel;
```

```
    unsigned T38RcvrControlRedundancyLevel;
int RTPRcvrClockSkew;
} FAX_RES;
```

RES Structure Parameters

The `RES` structure is used by many functions to return status information to the caller. In addition, the `RES`, `INFO_RES`, and `FAX_RES` structures are embedded in the `args_fax` structure.

The possible `res.status` values are:

BT_STATUS_OK Normal return.	0
BT_STATUS_ERROR Error return.	2
BT_STATUS_ERROR_DIAL Dialing error.	3
BT_STATUS_ERROR_HANGUP Hangup error.	4
BT_STATUS_USER_TERMINATED User function caused termination.	5
BT_STATUS_TIMEOUT Timeout occurred.	6
BT_STATUS_ALERT Alert occurred.	7

BT_STATUS_OK

In some cases the `res.line_status` value gives more information about the results. When the status value is `BT_STATUS_OK`, no associated `line_status` value exists for most functions. However, where noted, certain functions (for example, *BfvLineOriginateCall*) set the `line_status` value for `BT_STATUS_OK`.

BT_STATUS_ERROR

When the status value is `BT_STATUS_ERROR`, `line_status` gives more detailed information about the error that occurred. Possible `res.line_status` values for `BT_STATUS_ERROR` are:

APIERR_UNCLASSIFIED No further info provided.	0
APIERR_FILEIO File I/O error occurred.	1
APIERR_FILEFORMAT Bad file format.	2

APIERR_BOARDCAPABILITY Hardware or firmware does not support capability.	3
APIERR_NOTCONNECTED Channel not in proper state.	4
APIERR_BADPARAMETER Bad parameter value used.	5
APIERR_MEMORY Memory allocation error.	6
APIERR_BADSTATE The channel is not in a required state.	7
APIERR_TOOSOON Dialing was attempted too soon.	8
APIERR_BUSY Resource busy.	9
APIERR_DRV_OPEN_ERROR Driver open call failed, invalid channel or driver not correctly installed.	10
APIERR_DRV_IOCTL_ERROR Driver call failed.	11
APIERR_VERSION Incompatible driver version.	12
APIERR_INVALID_PORT Invalid port; faxinit probably not run correctly.	13
APIERR_RINGING Ringing during dialing attempt.	14
APIERR_INFOPKT_NESTING Indir infopkt nesting level too deep.	15
APIERR_MAX_TAGS Maximum number of TIFF tags exceeded.	16
APIERR_LOCK_FAILED An attempt to gain a lock failed.	17
APIERR_INSUFF_BUFFER Buffer size too small to receive data.	18
APIERR_INVALID_DEST_ADDR Destination address not found or invalid.	19
APIERR_PACKET_CREATION Packet or command creation error.	20
APIERR_PACKET_PARSE Packet or command parse error.	21

APIERR_PACKET_SEND 22
Packet send error.

APIERR_PACKET_RECEIVE 23
Packet receive error.

APIERR_DATA 24
DATA encountered during command processing.

APIERR_INVAL_BOARD_PARAM 25
Invalid parameter values received from firmware.

APIERR_FIRMWARE_ERR_DETECTED 26
Firmware detected an error.

APIERR_MODULE_REMOVED 27
Module was removed.

APIERR_BOARD_NO_RESPONSE 28
Board not responding.

APIERR_ASYNC_LP_ERR 29
Async_lp value error.

APIERR_ASYNC_CONTEXT_ERR 30
Async context error.

APIERR_DRV_RESOURCES 31
Driver out of resources.

APIERR_MODULE_RESET_FAILURE 32
Module reset failure.

APIERR_MODULE_I20_FAILURE 33
Module I20 enable failure.

APIERR_MODULE_CONFIG_TIMEOUT 34
No valid response for module configuration.

BT_STATUS_ERROR_DIAL

When the status value is `BT_STATUS_ERROR_DIAL`, `line_status` gives more detailed information about the dialing error that occurred. Possible `res.line_status` values for `BT_STATUS_ERROR_DIAL` are:

DIAL_OK 257
Dialing completed successfully.

DIAL_NO_DIAL_TONE 258
No dial tone detected.

DIAL_NO_LOOP_CUR 259
No loop current detected.

DIAL_LOCAL_IN_USE 260
Local phone in use successfully.

DIAL_TRUNK_BUSY 261
Busy trunk line detected.

DIAL_SLOT_BUSY 265
T1 time slot busy.

DIAL_CALL_COLLISION 266
Ringing detected during dialing.

DIAL_NO_WINK 267
2nd or later wink missing for Feature Group D.

Note: To determine the corresponding return value that the *BfvLineDialString* function returns, subtract `257` `DIAL_OK)` from each of the above `res.line_status` values for `BT_STATUS_ERROR_DIAL`. See *BfvLineDialString*, for more detailed information.

BT_STATUS_ERROR_HANGUP

When the status value is `BT_STATUS_ERROR_HANGUP`, `line_status` always contains a hangup code; see *Appendix , Hangup Codes on page 1371*.

BT_STATUS_TIMEOUT

Possible `res.line_status` values for `BT_STATUS_TIMEOUT` are:

API_TO_TIME 0
Timeout due to total time elapsed.

API_TO_SILENCE 1
Timeout due to silence timeout (speech record only).

Anytime the application uses a *BfvSpeechRecord*... function, and the function returns with a `res.status` value of `BT_STATUS_TIMEOUT`, the `res.line_status` value will indicate the type of timeout that occurred (total time or silence). Not all `res.status` or `res.line_status` values are possible with all functions.

Use the *BfvErrorMessage* function to get a textual description of the error stored in a `RES` structure.

CALL_RES Structure Parameters

The following functions use this structure to return status information while waiting to detect an incoming call.

- *BfvCallWaitForComplete*
- *BfvCallWaitForSetup*

call_type

A value indicating the type of incoming call detected. This field only returns the following value:

```
CALL_TYPE_ISDN
```

Note: Dialogic only maintains the *call_type* field for backward compatibility and does not recommend its use for applications developed with Brooktrout SDK 6.0 or later.

dest_id

A null-terminated ASCII string that identifies the captured DID digits.

called_party_subaddress

A null-terminated ASCII string that identifies the received called party subaddress. This value is only indicated for certain protocols.

calling_party_number

A null-terminated ASCII string that identifies the received calling party number (also known as caller ID).

calling_party_subaddress

A null-terminated ASCII string that identifies the received calling party subaddress.

When the call control configuration file (*callctrl.cfg*) has the *caller_id* parameter turned on for an analog port, the Bfv API returns the name of the caller in this field of the `CALL_RES` structure and also in the *name_ident* field (see *page 1354*).

redir_number

A null-terminated ASCII string that identifies the received redirection number or the destination number when diverting the call. The Bfv API only indicates this value for certain protocols (for example, QSIG).

redir_reason

A value identifying the reason for sending or receiving a destination number for a redirected or diverted call. This value is only indicated for certain protocols (for example, QSIG). Values are:

`BT_REDIR_UNKNOWN`
No redirection or unknown reason.

`BT_REDIR_CALL_FWD_BUSY`
Call forwarding busy or called Data Terminal Equipment (DTE) busy.

`BT_REDIR_CALL_FWD_NOANS`
Call forwarding no reply.

`BT_REDIR_OOS`
Called DTE out of service.

`BT_REDIR_CALL_FWD_DTE`
Call forwarding by called DTE.

`BT_REDIR_CALL_FWD_ALL`
Call forwarding unconditional or systematic call redirection.

`DIVERT_NONE`
Value used for call that does not divert.

`DIVERT_BUSY`
Call diverted for busy condition.

`DIVERT_UNCONDITIONAL`
Call diverted without conditions.

`DIVERT_NO_RESPONSE`
Call diverted for unresponsive line.

name_ident

A null-terminated ASCII string that identifies the name associated with the received call (also known as caller ID name). The field allows a maximum of 50 characters (`ECC_MAX_NAME_STR`)

When the call control configuration file (*callctrl.cfg*) has the *caller_id* parameter enabled for an analog port or an E1 or T1 port using the QSIG protocol, the Bfv API returns the name of the caller in this field of the `CALL_RES` structure.

name_char_set

Indicates the international standard specification (ISOxxx) of the character set in use. Values are:

`NAME_CHAR_SET_UNKNOWN` `-1`

Unknown character set in use.

`NAME_CHAR_SET_NOT_INCLUDED` `0`

Name does not identify a character set and the Bfv API does not send one.

`NAME_CHAR_SET_ISO8859_1` `1`

Indicates use of character set defined by ISO 8859-1 international standard.

`NAME_CHAR_SET_ISO8859_2` `3`

Indicates use of character set defined by ISO 8859-2 international standard.

`NAME_CHAR_SET_ISO8859_3` `4`

Indicates use of character set defined by ISO 8859-3 international standard.

`NAME_CHAR_SET_ISO8859_4` `5`

Indicates use of character set defined by ISO 8859-4 international standard.

`NAME_CHAR_SET_ISO8859_5` `6`

Indicates use of character set defined by ISO 8859-5 international standard.

`NAME_CHAR_SET_ISO8859_7` `7`

Indicates use of character set defined by ISO 8859-7 international standard.

`NAME_CHAR_SET_ISO10646_BMP` 8

Indicates use of character set defined by ISO 10646-1 and ITU-T Recommendation X.680 international standards.

`NAME_CHAR_SET_ISO10646_UTF` 9

Indicates use of character set defined by UTF-8-STRING Annex R in ISO 10646-1 international standard.

connected_num

Indicates a null-terminated string of up to 31 characters (`MAX_CONN_NUM`) that provides the telephone number of the connected party.

referred_id

A null-terminated ASCII string that identifies the referrer.

INFO_RES Structure Parameters

The *BFVFaxGetRemoteInfo* function uses this structure to report the remote ID and any NSF/NSS/NSC or SUB/PWD/SEP information from the remote fax device.

remote_id

A null-terminated ASCII string indicating the ID (CSI, TSI, or CIG) of the remote fax device. The remote fax provides its ID as a means of identification.

nss_nsf_frame

A parameter used to report any non-standard facilities FSK (NSF, NSS, or NSC) data received from the remote fax device. See the *BfvFaxGetRemoteInfo* function in *Volume 4* for the format of this data.

subaddress

A null-terminated ASCII string that contains a subaddress FSK (SUB) received from the remote fax device.

password

A null-terminated ASCII string that contains an FSK (PWD) password received from the remote fax device.

selective_polling

A null-terminated ASCII string that provides the contents of a selective polling FSK (SEP) received from the remote fax device.

PAGE_RES Structure Parameters

A linked list of PAGE_RES structures is generated by the Bfv API. One structure is defined for each page-complete interrupt. These structures are accessed through the reslist_head and reslist_tail fields of the FAX_RES structure. The FAX_RES structure is used by the *BfvFaxSend*, *BfvFaxReceive*, and *BfvFaxPoll* functions but, in other cases, can be set with the LINE_FAX_RES macro. For more information on the LINE_FAX_RES macro, see *Volume 4*.

The PAGE_RES structures contain information that is useful to an application, and the application must free this memory when it is no longer needed. If you do not want to use the linked list, set the *max_pagelist* field in the user-defined configuration file to 0.

page_complete_type

Defines the page break type. Two types are defined:

- Noncontinuing (2a) types are expected page breaks resulting from an end-of-page command sent to the channel by the Bfv API.
- Continuing (2b) types are unexpected page breaks generated by the channel during its ASCII text to G3 conversion process.

continue_breaks

Depends on the *page_complete_type* parameter.

- If the *page_complete_type* parameter is (2b), *continue_breaks* is 0 and is ignored.
- If the *page_complete_type* parameter is (2a), *continue_breaks* is the number of channel-generated page breaks (2b) since the last application-generated page break (2a).

ascii_bytes

The number of bytes of ASCII data converted to G3 format for this particular transmitted page. Currently returns 0.

bad_lines

The total number of error-containing G3 lines detected on this particular page, (maximum of 255 errors reported).

total_lines

The total number of G3 lines that the channel has transmitted or received for this particular page.

total_rcv

The total number of G3 lines that the host has received for this particular page. This value may differ from *total_lines*.

bit_rate

The bit rate at which this particular page was transmitted or received.

BITRATE_RSLT_2400_V27	0	2400 bps (V.27)
BITRATE_RSLT_4800_V27	1	4800 bps (V.27)
BITRATE_RSLT_7200_V29	2	7200 bps (V.29)
BITRATE_RSLT_9600_V29	3	9600 bps (V.29)
BITRATE_RSLT_7200_V17	4	7200 bps (V.17)
BITRATE_RSLT_9600_V17	5	9600 bps (V.17)
BITRATE_RSLT_12000_V17	6	12000 bps (V.17)
BITRATE_RSLT_14400_V17	7	14400 bps (V.17)
BITRATE_RSLT_12000_V33	8	12000 bps (V.33)
BITRATE_RSLT_14400_V33	9	14400 bps (V.33)
BITRATE_RSLT_2400_V34	10	2400 bps (V.34)
BITRATE_RSLT_4800_V34	11	4800 bps (V.34)
BITRATE_RSLT_7200_V34	12	7200 bps (V.34)
BITRATE_RSLT_9600_V34	13	9600 bps (V.34)
BITRATE_RSLT_12000_V34	14	12000 bps (V.34)
BITRATE_RSLT_14400_V34	15	14400 bps (V.34)
BITRATE_RSLT_16800_V34	16	16800 bps (V.34)
BITRATE_RSLT_19200_V34	17	19200 bps (V.34)
BITRATE_RSLT_21600_V34	18	21600 bps (V.34)
BITRATE_RSLT_24000_V34	19	24000 bps (V.34)
BITRATE_RSLT_26400_V34	20	26400 bps (V.34)
BITRATE_RSLT_28800_V34	21	28800 bps (V.34)
BITRATE_RSLT_31200_V34	22	31200 bps (V.34)
BITRATE_RSLT_33600_V34	23	33600 bps (V.34)

sig_level

The signal level at which this particular page was transmitted or received. Computed at the time of last training. Not valid for T.38 or V.34.

Typical values are:

-40 dBm	marginal
-35 dBm	weak
-30 dBm	acceptable
-25 dBm	good
-20 dBm	strong
-15 dBm	very strong

line_noise

The ambient line noise present when this particular page was transmitted or received. Computed at the time of last training. Not valid for T.38 or V.34.

Typical values are:

-65 dBm	extremely quiet
-60 dBm	very quiet
-55 dBm	quiet
-50 dBm	acceptable
-45 dBm	noisy
-40 dBm	almost unusable

sig_quality

The signal quality present when this particular page was received. Computed at the time of last training. Not valid for T.38 or V.34. Lower values (positive) are better. A value of 1 would represent an extremely good signal quality.

confirm_value

The FSK command sent by the receiving fax machine to the transmitting fax machine after a single-page transmission ends. For example, MCF is a typical confirmation value for a page with no errors, and RTN is a typical value for a page with many errors. Many symbols defined for use with FSK data are defined in a header file - *boston/driver/inc/fsk.h*.

direction

Contains 1 if this page was transmitted, or 0 if this page was received.

resolution

Contains a value indicating the resolution of the transmitted page. See the *BfvFaxBeginSendRaw* function for *resolution* values.

width

Contains a value indicating the width of the transmitted page.

See *BfvFaxBeginSendRaw* in *Volume 4* for *width* values.

eff_page_type

If nonzero, indicates that this page contained enhanced fax format data. The value indicates the specific type.

See *BfvFaxBeginSendRaw* in *Volume 4* for *eff_page_type* values.

*PAGE_RES *next*

Pointer to the next page result structure.

bad_lines_NS

Number of single bad lines (scaled by resolution).

bad_lines_N23

Number of 2-3 consecutive bad lines (scaled by resolution).

bad_lines_N4

Number of 4 or more consecutive bad lines (scaled by resolution).

num_ECM_frames

Number of ECM frames.

num_PPR

Total number of PPRs.

num_PPR_NS

Number of single PPRs.

num_PPR_N23

Number of 2-3 consecutive PPRs.

num_PPR_N4

Number of 4 or more consecutive PPRs.

time_t4

Maximum time to receive responses to commands, in ms.

time_training

Duration of training zeroes, in ms.

iaf_initial_speed

The initial IAF speed for the current page in bits per second.

iaf_final_speed

The final IAF speed for the current page in bits per second.

sig_level

The signal level at which this particular page was transmitted or received. Computed at the time of last training. Not valid for T.38 or V.34.

Typical values are:

-40 dBm	marginal
-35 dBm	weak
-30 dBm	acceptable
-25 dBm	good
-20 dBm	strong
-15 dBm	very strong

sig_quality

The signal quality present when this particular page was received. Computed at the time of last training. Not valid for T.38 or V.34. Lower values (positive) are better. A value of 1 would represent an extremely good signal quality.

line_noise

The ambient line noise present when this particular page was transmitted or received. Computed at the time of last training. Not valid for T.38 or V.34.

Typical values are:

-65 dBm	extremely quiet
-60 dBm	very quiet
-55 dBm	quiet
-50 dBm	acceptable
-45 dBm	noisy
-40 dBm	almost unusable

FAX_RES Structure Parameters

The Bfv API uses this structure to report information about the completed fax session. The Bfv API automatically allocates and stores PAGE_RES structures in a linked list within *args.fax_res*. The application must free these structures after use to release the memory.

number_of_pages

The total number of pages sent or received. These pages are the total of both noncontinuing and continuing types that are generated by the application, up to a maximum of *max_pagelist* (see *User-Defined Configuration File on page 1152*).

bad_pages

The number of pages that had bad lines or a confirmation value that indicated the page was received with bad lines.

remote_id

The identification of the remote fax machine. This field is used only when the application calls the *BfvFaxSend*, *BfvFaxReceive*, or *BfvFaxPoll* function.

*PAGE_RES *reslist_head*

Pointer to the beginning of the linked list of the PAGE_RES structure. This memory is available for reading by the application after fax transmission/reception is complete. The application must free this memory.

*PAGE_RES *reslist_tail*

Pointer to the end of the linked list of the PAGE_RES structure. This memory is available for reading by the application after fax transmission/reception is complete. The application must free this memory.

initial_bit_rate

Initial bit rate at which negotiation was first attempted. Same format as the bit_rate field of PAGE_RES.

ffom

Facsimile figure of merit (integer 1-7), defined by E.458 standard.

One of the FFOM_... definitions:

FFOM_COMPLETE_MAXSP_ERROR_FREE	1
FFOM_COMPLETE_MAXSP_ERRORED	2
FFOM_COMPLETE_MAXSP_SEV_ERRORED	3
FFOM_COMPLETE_NONMAXSP_ERROR_FREE	4
FFOM_COMPLETE_NONMAXSP_ERRORED	5
FFOM_COMPLETE_NONMAXSP_SEV_ERRORED	6
FFOM_INCOMPLETE	7

time_t1

Time for both fax devices to identify each other, in seconds. Rcv only.

time_t38

Time to switch into T.38, in ms.

termination_phase

The phase in which the call terminated.

One of the TERM_PHASE_... definitions:

TERM_PHASE_A	0
TERM_PHASE_B_PRE_MSG	1
TERM_PHASE_B_POST_MSG	2
TERM_PHASE_C	3
TERM_PHASE_D	4
TERM_PHASE_E	5

count_RTP

Count of RTPs (non-ECM).

count_RTN

Count of RTNs (non-ECM).

count_CRP

Count of CRPs.

count_CTC

Count of CTCs.

TRxMediaType media_type

The media type of the call. One of the MEDIA_TYPE_... definitions:

MEDIA_TYPE_NONE 1

MEDIA_TYPE_T38 2

MEDIA_TYPE_RTP 3

RTPRcvrPackets

Number of received RTP packets.

RTPSenderPackets;

Number of sent RTP packets.

RTPRcvrLostPackets

Number of lost/late RTP packets.

RTPRcvrLostPackets_NS

Number of single lost packets.

RTPRcvrLostPackets_N23

Number of 2-3 consecutive lost packets.

RTPRcvrLostPackets_N4

Number of 4 or more consecutive lost packets.

RTPRcvrJitter

Average jitter.

RTPRcvrOOSPackets

Out of Sequence Packets.

RTPRcvrRedundancyLevel

Level of redundancy used for received packets.

T38RcvrPackets

Number of T.38 packets received.

T38RcvrOctets

Number of T.38 octets received.

T38SenderPackets

Number of T.38 packets sent.

T38SenderOctets

Number of T.38 octets sent.

T38RcvrLostPackets

Number of T.38 packets lost and not recovered on receive.

T38RcvrLostPackets_NS

Number of single lost and not recovered packets.

T38RcvrLostPackets_N23

Number of 2-3 consecutive lost and not recovered packets.

T38RcvrLostPackets_N4

Number of 4 or more consecutive lost and not recovered packets.

T38RcvrLostOctets

Number of T.38 octets lost on receive.

T38RcvrOOSPackets

Out of Sequence Packets.

T38RcvrUsedRedundacy

Number of times redundant packets were required.

T38RcvrImageRedundancyLevel

Level of redundancy used for received image packets. Rcv only.

T38RcvrControlRedundancyLevel

Level of redundancy used for received control packets.

int RTPRcvrClockSkew

RTP clock skew is calculated with the following equation, which makes use of the least squares estimator with a single predictor value. T_i is the local clock time when packet i arrives, S_i is the RTP packet timestamp of packet i, "n " is the total number of RTP packets received, and D is the duration of the RTP session.

$$Skew\ Time = \left(\left(\frac{\left(\sum_{i=1}^{n} T_i * S_i\right) * (n-1)}{\left(n * \sum_{i=1}^{n} T_i\right) * \left(\sum_{i=1}^{n} T_i\right)^2}\right) - 1\right) * D$$

The least squares estimator result is the slope equal to the correlation between the local clock timestamps of the arriving RTP packets and the corresponding RTP header timestamps. A slope value that is greater than one indicates that the remote end's clock is faster than the local clock. A slope value that is less than one indicates a slower remote clock. Subtracting one from the calculated slope and multiplying the result by the duration of the RTP session, yields the total time contributed by the skew in microseconds.

DCS and DIS/DTC Info Structures

These structures contain information decoded from the DCS, DIS, and DTC FSK messages sent between the transmitter and receiver. The structures are defined and fully documented in the *dcs.h* header file. The application program can access them with the `LINE_DCS` and `LINE_DIS_DTC` macros (see *Macros* in *Volume 4* for more detailed information on these macros).

```
struct dcs_info {
    unsigned char sent;
    unsigned char Receiver_t4;
    unsigned char Data_Signalling_Rate;
    unsigned char Resolution;
    unsigned char TwoD_Coding;
    unsigned char Recording_Width;
    unsigned char Max_Recording_Length;
    unsigned char Min_Scan_Line_Time;
    unsigned char Handshake_2400_bit;
    unsigned char Uncompressed_Mode;
    unsigned char Error_Correction_Mode;
    unsigned char Error_Limiting_Mode;
    unsigned char MMR;
    unsigned char Resolution_Unit;
    unsigned char Binary_File_Transfer;
    unsigned char Document_Transfer_Mode;
    unsigned char Edifact_Transfer;
    unsigned char Basic_Transfer_Mode;
    unsigned char Character_Mode;
    unsigned char Mixed_Mode;
    unsigned char Processable_Mode_26;
    unsigned char Store_And_Fwd_Internet_Fax;
    unsigned char Real_Time_Internet_Fax;
    unsigned char Lossless_Mode;
    unsigned char Plane_Interleave_Mode;
    unsigned char ADPCM;
```

```
    unsigned char Digital_Network;
    unsigned char FullDuplex;
    unsigned char JPEG_Coding;
    unsigned char JPEG_FullColor;
    unsigned char JPEG_Default_Tables;
    unsigned char JPEG_12Bit;
    unsigned char JPEG_NoSubSampling;
    unsigned char JPEG_CustomIlluminant;
    unsigned char JPEG_CustomGamut;
    unsigned char NA_Letter;
    unsigned char NA_Legal;
    unsigned char JBIG_Coding;
    unsigned char JBIG_L0;
    unsigned char HKM_Key_Mgt;
    unsigned char RSA_Key_Mgt;
    unsigned char Override_Mode;
    unsigned char HFX40_Cipher;
    unsigned char HFX40_Hashing;
    unsigned char MRC_Mode;
    unsigned char MRC_Page_Length_Strips;
    unsigned char PhaseC_BFT_Negotiations;
    unsigned char IRA;
};

struct dis_dtc_info {
    unsigned char sent;
    unsigned char Transmitter_t4;
    unsigned char Receiver_t4;
    unsigned char Data_Signalling_Rate;
    unsigned char Resolution;
    unsigned char TwoD_Coding;
    unsigned char Recording_Width;
    unsigned char Max_Recording_Length;
    unsigned char Min_Scan_Line_Time;
    unsigned char Handshake_2400_bit;
```

```
unsigned char Uncompressed_Mode;
unsigned char Error_Correction_Mode;
unsigned char Error_Limiting_Mode;
unsigned char MMR;
unsigned char Resolution_200H_400V;
unsigned char Resolution_300H_300V;
unsigned char Resolution_400H_400V;
unsigned char Resolution_Unit;
unsigned char Min_Scan_Higher;
unsigned char Selective_Polling;
unsigned char Subaddressing_Capability;
unsigned char Password_Capability;
unsigned char Data_File_Capable;
unsigned char Binary_File_Transfer;
unsigned char Document_Transfer_Mode;
unsigned char Edifact_Transfer;
unsigned char Basic_Transfer_Mode;
unsigned char Character_File_Capable;
unsigned char Character_Mode;
unsigned char Mixed_Mode;
unsigned char Processable_Mode_26;
unsigned char Store_And_Fwd_Internet_Fax;
unsigned char Real_Time_Internet_Fax;
unsigned char V8_Capability;
unsigned char Num_Preferred_Octets;
unsigned char Mult_Selective_Polling;
unsigned char Polled_SubAddress;
unsigned char Lossless_Mode;
unsigned char Plane_Interleave_Mode;
unsigned char ADPCM;
unsigned char Digital_Network;
unsigned char FullDuplex;
unsigned char JPEG_Coding;
unsigned char JPEG_FullColor;
unsigned char JPEG_12Bit;
```

```
    unsigned char JPEG_NoSubSampling;
    unsigned char JPEG_CustomIlluminant;
    unsigned char JPEG_CustomGamut;
    unsigned char NA_Letter;
    unsigned char NA_Legal;
    unsigned char JBIG_Coding;
    unsigned char JBIG_L0;
    unsigned char HKM_Key_Mgt;
    unsigned char RSA_Key_Mgt;
    unsigned char Override_Mode;
    unsigned char HFX40_Cipher;
    unsigned char HFX40_Hashing;
    unsigned char MRC_Mode;
    unsigned char MRC_Page_Length_Strips;
    unsigned char ColorRes_300x300_400x400;
    unsigned char ColorRes_100x100;
    unsigned char PhaseC_BFT_Negotiations;
    unsigned char ISP;
    unsigned char IRA;
    unsigned char Resolution_600H_600V;
    unsigned char Resolution_1200H_1200V;
    unsigned char Resolution_300H_600V;
    unsigned char Resolution_400H_800V;
    unsigned char Resolution_600H_1200V;
};
```

C - Hangup Codes

This appendix explains the codes returned when a disconnect occurs.

Hangup codes identify disconnections that have occurred and the reasons for these. For example, these codes can be returned because a loss of loop current or a serious error occurred that the Bfv API needs to report to the application.

When:

```
res.status = BT_STATUS_ERROR_HANGUP
```

`res.line_status` contains a hangup code. The hangup codes are grouped according to classification (usually by T.30 protocol phase). The code values are in decimal format.

This appendix explains the code types as follows:

- *Call Placement Codes on page 1373*
- *Transmit Phase A Codes on page 1373*
- *Transmit Phase B Codes on page 1374*
- *Transmit Phase D Codes on page 1376*
- *Receive Phase B Codes on page 1380*
- *Receive Phase D Codes on page 1382*
- *Phase C Codes on page 1383*
- *Miscellaneous Codes on page 1384*
- *Bfv API-Created Codes on page 1385*

The fax protocol itself has a diverse set of precisely identified failure reasons. The ITU-T (previously CCITT) T.30 specification describes this fax protocol, and you should obtain a copy of it to gain a better understanding of both the protocol and the failure conditions that generate the hangup codes listed and described in this appendix. You can obtain a copy of ITU-T T.30 fax protocol from:

http://www.itu.int

Call Placement Codes

Value	0
Hangup Code	HNG_NORMAL_XMIT
Description	Normal and proper end of connection. While this value is considered by the firmware to be a successful fax transmit result, if it occurs in conjunction with `BT_STATUS_ERROR_HANGUP`, it still indicates that an error has occurred.

Value	1
Hangup Code	HNG_RNG_DET
Description	Ring detected without a successful handshake.

Value	2
Hangup Code	HNG_ABORT
Description	Call Aborted.

Value	3
Hangup Code	HNG_NO_LOOP_CURRENT
Description	No loop current or A/B signaling bits.

Value	4
Hangup Code	HNG_ISDN_DISCONNECT
Description	ISDN disconnection.

Transmit Phase A Codes

Value	11
Hangup Code	HNG_T1_TIMEOUT
Description	No answer, T.30 T1 timeout.

Transmit Phase B Codes

Value	5
Hangup Code	HNG_INVAL_POLL_ATT
Description	Invalid Polling Attempt

Value	20
Hangup Code	HNG_XMITB_TIMEOUT
Description	Unspecified transmit Phase B error.

Value	21
Hangup Code	HNG_XMITB_NORM
Description	Remote cannot receive or send.

Value	22
Hangup Code	HNG_XMITB_MISC
Description	COMREC error, Phase B transmit.

Value	23
Hangup Code	HNG_XMITB_COMREC_VCNR
Description	COMREC invalid command received.

Value	24
Hangup Code	HNG_XMITB_SE
Description	RSPREC error.

Value	25
Hangup Code	HNG_XMITB_DCS_FTC
Description	DCS sent three times without response.

Value	26
Hangup Code	HNG_XMITB_DIS_FTC
Description	DIS/DTC received three times; DCS not recognized.

Value	27
Hangup Code	HNG_XMITB_TRAINFAIL
Description	Failure to train.

Value	28
Hangup Code	HNG_XMITB_RSPREC_VCNR
Description	RSPREC invalid response received.

Value	29
Hangup Code	HNG_XMITB_COMREC_DCN
Description	DCN received in COMREC.

Value	30
Hangup Code	HNG_XMITB_RSPREC_DCN
Description	DCN received in RSPREC.

Value	33
Hangup Code	HNG_PHASEB_INCOMPAT_FMT
Description	Incompatible fax formats, for example, a page width mismatch.

Value	34
Hangup Code	HNG_XMITB_INVAL_DMACNT
Description	Invalid DMA count specified for transmitter.

Value	35
Hangup Code	HNG_XMITB_FTM_NOECM
Description	Binary File Transfer specified, but ECM not enabled on transmitter.

Value	36
Hangup Code	HNG_XMITB_INCMP_FTM
Description	Binary File Transfer mode specified, but not supported by receiver.

Value	37
Hangup Code	HNG_XMITB_INCMP_EFF
Description	Remote does not support EFF page options required by host.
Value	38
Hangup Code	HNG_XMITB_NOEFF
Description	Remote does not support EFF page coding.

Transmit Phase D Codes

Value	40
Hangup Code	HNG_XMITD_RR_NORES
Description	No response to RR after three tries.

Value	41
Hangup Code	HNG_XMITD_CTC_NORES
Description	No response to CTC, or response was not CTR.

Value	42
Hangup Code	HNG_XMITD_T5TO_RR
Description	T5 time out since receiving first RNR.

Value	43
Hangup Code	HNG_XMITD_NOCONT_NSTMSG
Description	Do not continue with next message after receiving ERR.

Value	44
Hangup Code	HNG_XMITD_ERRRES_EOREOP
Description	ERR response to EOR-EOP or EOR-PRI-EOP.

Value	45
Hangup Code	HNG_XMITD_RTN_DCN
Description	Transmitted DCN after receiving RTN.

Value	46
Hangup Code	HNG_XMITD_PPR_EOR
Description	EOR-MPS, EOR-EOM, EOR-NULL, EOR-PRI-MPS, or EOR-PRI-EOM sent after fourth PPR received.

Value	51
Hangup Code	HNG_XMITD_SE
Description	RSPREC error.

Value	52
Hangup Code	HNG_XMITD_MPS_FTC
Description	No response to MPS, repeated three times.

Value	53
Hangup Code	HNG_XMITD_MPS_VCNR
Description	Invalid response to MPS.

Value	54
Hangup Code	HNG_XMITD_EOP_FTC
Description	No response to EOP repeated three times.

Value 55
Hangup Code HNG_XMITD_EOP_VCNR
Description Invalid response to EOP.

Value 56
Hangup Code HNG_XMITD_EOM_FTC
Description No response to EOM, repeated three times.

Value 57
Hangup Code HNG_XMITD_EOM_VCNR
Description Invalid response to EOM.

Value 60
Hangup Code HNG_XMITD_RSPREC_DCN
Description DCN received in RSPREC.

Value 61
Hangup Code HNG_XMITD_PPSNULL_NORES
Description No response received after third try for PPS-NULL.

Value 62
Hangup Code HNG_XMITD_PPSMPS_NORES
Description No response received after third try for PPS-MPS.

Value 63
Hangup Code HNG_XMITD_PPSEOP_NORES
Description No response received after third try for PPS-EOP.

Value 64
Hangup Code HNG_XMITD_PPSEOM_NORES
Description No response received after third try for PPS-EOM.

Value	65
Hangup Code	HNG_XMITD_EORNULL_NORES
Description	No response received after third try for EOR-NULL.

Value	66
Hangup Code	HNG_XMITD_EORMPS_NORES
Description	No response received after third try for EOR-MPS.

Value	67
Hangup Code	HNG_XMITD_EOREOP_NORES
Description	No response received after third try for EOR-EOP.

Value	68
Hangup Code	HNG_XMITD_EOREOM_NORES
Description	No response received after third try for EOR-EOM.

Receive Phase B Codes

Value	5
Hangup Code	HNG_INVAL_POLL_ATT
Description	Invalid Polling Attempt

Value	70
Hangup Code	HNG_RCVB_TIMEOUT
Description	Unspecified receive Phase B error.

Value	71
Hangup Code	HNG_RCVB_SE
Description	RSPREC error.

Value	72
Hangup Code	HNG_RCVB_MISC
Description	COMREC error.

Value	73
Hangup Code	HNG_T2_PNOTREC
Description	T.30 T2 timeout, expected page not received.

Value	74
Hangup Code	HNG_RCVB_T1_TIMEOUT
Description	T.30 T1 timeout after EOM received.

Value	75
Hangup Code	HNG_NORMAL_RCV
Description	DCN received in COMREC. While this value is considered by the firmware to be a successful fax receive result, if it occurs in conjunction with `BT_STATUS_ERROR_HANGUP`, it still indicates that an error has occurred.

Value	76
Hangup Code	HNG_RCVB_RSPREC_DCN
Description	DCN received in RSPREC.

Value	77
Hangup Code	HNG_T2_TIMEOUT
Description	T.30 T2 timeout, expected page received.

Value	78
Hangup Code	HNG_RCVB_INVAL_DMACNT
Description	Invalid DMA count specified for receiver.

Value	79
Hangup Code	HNG_RCVB_FTM_NOECM
Description	Binary File Transfer specified, but ECM not supported by receiver.

Receive Phase D Codes

Value	101
Hangup Code	HNG_RCVD_SE_VCNR
Description	RSPREC invalid response received.

Value	102
Hangup Code	HNG_RCVD_COMREC_VCNR
Description	COMREC invalid response received.

Value	103
Hangup Code	HNG_RCVD_T3TO_NORES
Description	T3 timeout; no local response for remote voice interrupt.

Value	104
Hangup Code	HNG_RCVD_T2TO
Description	T2 timeout; no command received after responding RNR.

Value	105
Hangup Code	HNG_RCVD_DCN_COMREC
Description	DCN received for command received.

Value	106
Hangup Code	HNG_RCVD_COMREC_ERR
Description	Command receive error.

Value	107
Hangup Code	HNG_RCVD_BLKCT_ERR
Description	Receive block count error in ECM mode.

Value	108
Hangup Code	HNG_RCVD_PGCT_ERR
Description	Receive page count error in ECM mode.

Value	109
Hangup Code	HNG_RCVD_EOR
Description	EOR received in phase D.

Value	110
Hangup Code	HNG_RCVD_RNRTO
Description	Timeout while repeating RNR.

Phase C Codes

Value	150
Hangup Code	HNG_RCVC_EOL_TIMEOUT
Description	No EOL received in a 5-second period.

Value	151
Hangup Code	HNG_RCVC_BAD_MMR
Description	Bad MMR data received from remote.

Value	152
Hangup Code	HNG_RCVC_ECM_ZERO_LINES
Description	Zero lines received from remote in ECM mode.

Miscellaneous Codes

Value	240
Hangup Code	HNG_INTERRUPT_ACK
Description	No interrupt acknowledge, timeout.

Value	241
Hangup Code	HNG_COMM_FAULT
Description	Loop current still present while playing recorder tone after timeout.

Value	242
Hangup Code	HNG_T30_HOLDUP
Description	T.30 holdup timeout.

Value	243
Hangup Code	HNG_HOLDUP_DCN
Description	DCN received from host in receive holdup section for FAX PAD mode.

Value	244
Hangup Code	HNG_HOLDUP_DCN_NON_FPAD
Description	DCN received from host in receive holdup section for non-FAX PAD mode.

Bfv API-Created Codes

Value	500
Hangup Code	HNG_ERROR_INTERRUPT
Description	An error interrupt occurred, indicating a problem with the channel too severe to continue. The value of the error interrupt can be obtained with the LINE_ERROR_INTR macro.

Value	501
Hangup Code	HNG_INTERRUPT_OVERRUN
Description	The application was unable to process incoming interrupts/commands fast enough, and information was lost. See LINE_INTR_OVERRUN in *Macros* section of *Volume 1, Chapter 6*.

Value	502
Hangup Code	HNG_UNEXPECTED_IRSDONE
Description	The channel generated an unexpected 03 (reset done) or 7F interrupt, indicating the existence of a firmware or hardware problem.

Value	503
Hangup Code	HNG_IOCTL_ERROR
Description	An Bfv API command to the driver returned an error value, indicating that the driver or the operating system detected an error.

Value	504
Hangup Code	HNG_OVERLAY_DLOAD_ERR
Description	Error reported at termination of fax overlay download.

Value	505
Hangup Code	HNG_MAX_TIMEOUT
Description	Maximum timeout exceeded. This code occurs when the user configuration file parameter *max_timeout* has been enabled and the specified timeout has expired.

D - BSMI and ISDN Cause Codes

This appendix defines and lists the BSMI and ISDN cause codes.

It has the following sections:

- *Defining BSMI Cause Codes*
- *Defining ISDN Cause Codes*

Defining BSMI Cause Codes

The cause data structure defined in *Volume 5, Chapter 2*, provides information relating to the source and reason for generation of a certain ISDN message. The Cause IE consists of:

- Coding standard in use
- Location of the equipment generating the message
- Additional diagnostic information

Table 28 contains the listing of possible cause values encountered by Dialogic during product testing and is provided here and in *IISDN.h* to assist application developers.

The Cause Data structure is used in the following messages:

Call Control

- *L4L3mCLEAR_REQUEST*
- *L3L4mCLEAR_REQUEST*
- *L3L4mDISCONNECT*

Supplemental

- *L4L3mSUSPEND_REJECT*
- *L4L3mRESUME_REJECT*
- *L3L4mSUSPEND_REJECT*
- *L3L4mRESUME_REJECT*

Table 28. BSMI Cause Codes

Decimal Value	Mnemonic	Message Generated Because
0	IISDNcausDEFAULT	Default cause or value not available.
1	IISDNcausUNALOC_NUM	The number was unallocated (unassigned).
2	IISDNcausNO_ROUTE	No route was specified to the Transit network.
6	IISDNcausBAD_CHAN	The channel specified was unacceptable.
16	IISDNcausNORML_CLR	Part of normal call clearing procedures.
17	IISDNcausUSER_BUSY	Called party was busy.
18	IISDNcausNO_USE_RSP	Message generated because no user responded.
21	IISDNcausCALL_REJ	Message generated because the call was rejected.
22	IISDNcausNUM_CHANGED	The called party number changed.
27	IISDNcausDEST_OOO	The destination was out of order.
28	IISDNcausINVALID_NUM	The number was in an invalid format.
29	IISDNcausFACIL_REJ	A facility reject.
30	IISDNcausSTAT_RESP	A response to a STATUS enquiry.
31	IISDNcausNRML_UNSPEC	Normal or unspecified reason.
34	IISDNcausNO_CHAN_AVL	No circuit/channel was available.
38	IISDNcausNET_OOS	The network was out of order.
41	IISDNcausTEMP_FAILURE	A temporary failure.
42	IISDNcausSW_CONJEST	Switching equipment congestion.
43	IISDNcausACCESS_DISC	The access information was discarded.
44	IISDNcausCKT_NOT_AVAIL	The requested circuit or channel was not available.

Table 28. BSMI Cause Codes (Continued)

Decimal Value	Mnemonic	Message Generated Because
45	IISDNcausPREEMPT	Preempted (AT&T special defined).
47	IISDNcausRES_UNAVAIL	The resource was unavailable or unspecified.
50	IISDNcausFAC_NOT_SUSC	The requested facility was not subscribed.
52	IISDNcausOUT_BARRED	Outgoing calls are not permitted.
54	IISDNcausIN_BARRED	Incoming calls are not permitted.
57	IISDNcausBR_CAP_AUTH	The requested bearer capability is not authorized.
58	IISDNcausBR_CAP_NA	The requested bearer capability is not available.
63	IISDNcausSRVC_NA	The service or option is not available or unspecified.
65	IISDNcausBR_SVC_NIMP	The bearer service is not implemented.
66	IISDNcausCHNTYP_NIMP	The channel type is not implemented.
69	IISDNcausFAC_NIMP	The requested facility is not implemented.
70	IISDNcausBR_REST_ONLY	Only restricted digital data is available for the bearer capability.
79	IISDNcausSVC_NIMP	The requested service is not implemented or unspecified.
81	IISDNcausINV_CALL_REF	An invalid call reference value was used.
82	IISDNcausCHAN_DNE	The identified channel does not exist.
88	IISDNlcausINCOMPAT_DST	An incompatible destination.
95	IISDNcausINVL_MSG	The transmitted Q.931 message was invalid.
96	IISDNcausMAND_IE	Mandatory Information Element missing.
97	IISDNcausMSGTYP_BAD	The message type does not exist or is not implemented.
98	IISDNcausINCOMPAT_MSG	The message was incompatible for the state of call.
99	IISDNcaus IE_NOT_EXIST	The transmitted Q.931 message included an IE that does not exist.

Table 28. BSMI Cause Codes (Continued)

Decimal Value	Mnemonic	Message Generated Because
100	IISDNcausINVL_IE_CONT	Invalid content in the Information Element.
101	IISDNcausMSG_COMPAT	Q.931 message which was transmitted was not compatible with the call state during which it was sent.
102	IISDNcausTIMER_EXPIR	Recovery on timer expiration.
111	IISDNcausPROTO_ERR	Protocol error.
127	IISDNcausINTERWORK	Interworking or an unspecified reason.

Defining ISDN Cause Codes

ISDN cause codes can be used as input or output arguments (*args.cause* or *args.cause_code*) when calling the following call control functions:

Function[1]	Input	Output
BfvCallDisconnect	*args.cause*	
BfvCallReject	*args.cause*	
BfvCallWaitForAccept		*args.cause*
BfvCallWaitForComplete		*args.cause*
BfvCallWaitForRelease		*args.cause*
BfvLineAnswer		*args.cause_code*
BfvLineTerminateCall		*args.cause_code*

1. Function details found in *Volume 2*.

Some cause codes are grouped by class number, see:

- *Table 29 on page 1391*
- *Table 30 on page 1392*
- *Table 31 on page 1392*
- *Table 32 on page 1393*
- *Table 33 on page 1393*
- *Table 34 on page 1395*
- *Table 35 on page 1396*
- *Table 36 on page 1396*
- *Table 37 on page 1396*

Tables 29 through 37 provide *Hex* and *Value* columns along with the description and meaning of each of the ISDN cause codes. The *Hex* column is the information contained in the Cause Information Element (IE) used in the Q.931 messages. This number is generated by logically ORing a 0x80 value with the hexadecimal conversion of the *Value* column. The *Value* column indicates the decimal representation of the ISDN cause code as defined in the ITU-T Q.850 specification.

Table 29. Class 000 - Normal Events

Hex	Value	Description	Meaning
81	1	Unallocated number	Indicates that the requested destination, although valid, cannot be reached.
82	2	No route to specified network	Sending equipment (sending the cause) is requested to route call through an unrecognized transit network.
83	3	No route to destination	Called user cannot be reached because the network does not serve the destination.
86	6	Channel unacceptable	The last identified channel is not acceptable to the sending entity.
87	7	Call awarded	Incoming call is connected to a channel already established for similar calls (for example: packet-mode X.25 virtual calls).
90	16	Normal call clearing	Call is cleared at the request of one of the users involved.
91	17	User busy	Called user cannot accept another call although compatibility is established.
92	18	No user responding	When a user does not respond to call establishment messages with either an alerting or connect indication within the allowed time.
93	19	User alerted, no answer	User provided an alerting indication but has not provided a connect indication within the allowed time.
95	21	Call rejected	Equipment sending the cause does not want to accept this call even though the equipment is not busy or incompatible.
96	22	Number changed	Indicates called party number is not assigned.
9A	26	Nonselected user clearing	User not awarded the incoming call.
9B	27	Destination out of order	Destination interface is not functioning correctly.
9C	28	Invalid number format	Called party number is invalid or incomplete.
9D	29	Facility rejected	Network cannot provide the facility requested.
9E	30	Response to STATus ENQuiry	The reason for generating the STATUS message was the prior receipt of a STATUS ENQUIRY message.
9F	31	Normal, unspecified	Used to report normal events only when no other cause in the normal class applies.

Table 30. Class 010 - Network Congestion

Hex	Value	Description	Meaning
A2	34	No channel available	An appropriate channel is not currently available to handle the call.
A3	35	Call queued (AT&T)	Network is not functioning. Immediate redial is unlikely to succeed.
A6	38	Network out of order	Network is not functioning. Immediate redial is unlikely to succeed.
A9	41	Temporary failure	Network is not functioning. Immediate redial is unlikely to succeed.
AA	42	Switching equipment congestion	Switching equipment generating this cause is experiencing a period of high traffic. AB 42 user information is discarded. The network cannot deliver access information to the remote user as requested. For example: ■ User-to-user information ■ Low-layer compatibility ■ Sub-address as indicated in the diagnostic The particular type of discarded access information is optionally included in the diagnostic.
AC	44	Requested channel not available	The channel indicated by the requesting entity cannot be provided by the other side of the interface.
AF	47	Resource unavailable, unspecified	A resource unavailable event only when no other cause in the resource unavailable class applies.

Table 31. Class 011 - Service or Option Not Available

Hex	Value	Description	Meaning
B1	49	Quality of service unavailable	Throughput or transit delay cannot be supported. The Quality of Service (as defined in Recommendation X.213) cannot be provided.
B2	50	Requested facility not subscribed	Requested supplementary service not provided by the network because the user has not completed the necessary administrative arrangements with its supporting networks.
B4	52	Outgoing calls barred (AT&T)	Outgoing calls are not permitted.

Table 31. Class 011 - Service or Option Not Available (Continued)

Hex	Value	Description	Meaning
B6	54	Incoming calls barred	Incoming calls are not permitted.
B9	57	Bearer capability not authorized	User is trying to make unauthorized use of equipment providing a bearer capability.
BA	58	Bearer capability not presently available	User has requested a bearer capability that is implemented by the equipment generating the cause, but is not available at this time.
BF	63	Service or option not available, unspecified	A service or option not available event only when no other cause in the service or option not available class applies.

Table 32. Class 100 - Service or Option Not Implemented

Hex	Value	Description	Meaning
C1	65	Bearer capability not implemented	Equipment sending this cause does not support the requested bearer capability.
C2	66	Channel type not implemented	Equipment sending this cause does not support the requested channel type.
C5	69	Requested facility not implemented	Equipment sending this cause does not support the requested supplementary service.
C6	70	Only restricted digital bearer capability available	Request for an unrestricted bearer service, but the equipment sending this cause only supports the restricted version.
CF	79	Service or option not implemented, unspecified	A service or option not implemented event only when no other cause in the service or option not implemented class applies.

Table 33. Class 101 - Invalid Message

Hex	Value	Description	Meaning
D1	81	Invalid call reference value	A message with a call reference that is not currently in use on the user network interface, received by the equipment sending the cause.
D2	82	Channel does not exist	Equipment sending this cause received a request to use a channel not activated on the interface for a call.

Table 33. Class 101 - Invalid Message (Continued)

Hex	Value	Description	Meaning
D3	83	Suspended call exists, call identity does not	A call resume attempted with a call identity that differs from that in use for any currently suspended call.
D4	84	Call identity in use	Network received a call suspended request. The request contained a call identity (including the null call identity) that is already in use for a suspended call within the domain of interfaces over which this call can be resumed.
D5	85	Invalid digit value for number	Network received a call resume request. The request contained a call identity information element that does not indicate any suspended call within the domain of interfaces over which the call can be resumed.
D6	86	Call having the requested call identity is cleared	The network has received a call resume request. This request contained a call identity information element that once indicated a suspended call; the suspended call was cleared while suspended (either by network timeout, or by a remote user).
D8	88	Incompatible destination	Equipment sending this cause received a request to establish a call that has low layer compatibility, high layer compatibility attributes (for example, data rate) that cannot be handled.
DB	91	Transit network does not exist	Incorrect format received for transit network identification (format defined in Annex C/Q.931).
DF	95	Invalid message, unspecified	Invalid message event only when no other cause in the invalid message call applies.

Table 34. Class 110 - Protocol Error

Hex	Value	Description	Meaning
E0	96	Mandatory information element is missing	Equipment sending this cause received a message that is missing an information element that must be present in the message before that message can be processed.[1]
E1	97	Message type nonexistent or not implemented	Equipment sending this cause received a message with a message type it does not recognize: Undefined message. Defined but not implemented by the equipment sending the cause.
E2	98	Message not compatible with call state or message type nonexistent or not implemented	Equipment sending this cause received a message that it considers non-permissible while in the call state; or a STATUS message received indicating an incompatible call state.
E3	99	Information element nonexistent or not implemented	Equipment sending this cause received a message that includes information elements not recognized because the information element identifier is not defined, or is defined but not implemented by the equipment sending the cause. However, the information element is not required to be present in the message to enable the equipment sending the cause to process the messages.[1]
E4	100	Invalid information element contents	Equipment sending this cause received an information element that it has implemented. However, the sending equipment was not able to implement the code because one or more of the fields were incorrectly coded.[1]
E5	101	Message not compatible with call state	The received message is incompatible with the call state.
E6	102	Recovery on timer expiry	A timer expired and an associated Q.931 error handling procedure is initiated.
EF	111	Protocol error, unspecified	An error event only when no cause in the protocol error class applies.

1. The particular Information Element is identified in the diagnostic byte. For example 81 E0 04 means that the bearer capability is not included by the PABX (Private Network) in the SETUP message. 0x04 is the Bearer Capability Information Element Identifier as specified in the standards.

Table 35. Class 111 - Inter-networking

Hex	Value	Description	Meaning
FF	127	Interworking unspecified	Interworking with a network that does not provide cause codes for its actions. Therefore, the precise cause for transmitting a message is unknown.

Table 36. Dialogic (formerly Brooktrout) Proprietary ISDN Cause Codes Returned by High-Level Call Control Functions

Hex	Value	Name	Meaning
3E8	1000	`CLEARcausNO_DIALTONE`	No dial tone detected when placing a call.
3E9	1001	`CLEARcausINTERNAL_DIAL_ERROR`	An internal dialing error has occurred.
3EA	1002	`CLEARcausNO_LOOPCURRENT`	No loop current detected.
3EB	1003	`CLEARcausJATE_REJECT`	Call failed to connect within the limits defined for the JATE standard's redial restriction.

Table 37. Diagnostic Byte

Hex	Value	Description
02	2	Transit network identity or network specific facility Information Element Identifier.
16	22	New destination number.
1D	29	Facility identification.
2B	43	Discarded Information Element Identifier.
2F	47	Information Element Identifier.
39	57	Attributes of bearer capability.
3A	58	Attributes of bearer capability.
41	65	Attributes of bearer capability.
42	66	Channel type.
58	88	Incompatible parameter.
5F	95	Message type.
60	96	Information Element Identifier.

Table 37. Diagnostic Byte (Continued)

Hex	Value	Description
61	97	Message type.
62	98	Message type.
63	99	Information Element Identifier.
64	100	Information Element Identifier.
65	101	Message type.
66	102	Timer number.

E - Infopkt Parameter Values

This appendix describes the voice and fax infopkt parameters.

It has the following sections:

- *Voice Infopkt Parameters*
- *Fax Infopkt Parameters*

All infopkts consist of a four-byte header that precedes some data. The header defines the infopkt's type and indicates its total length.

In each infopkt, define the header using `struct infopkt hdr`. It has the following format and parameters:

```
struct infopkt hdr
{
    unsigned short type;
    unsigned short length;
};
```

The following describes each of these parameters:

Parameter	*type*
Units	None.
Range	See the *infopkt.h* file for a complete list of infopkt types.
Default	None.

Parameter	*length*
Units	Bytes.
Range	Total length of the infopkt, including header and data. Maximum 30,000 bytes; recommend 1K limit.
Default	None.

Voice Infopkt Parameters

The voice infopkt parameters include the following:

- *End-of-Speech Parameter Infopkt on page 1400*
- *Prompt Map Infopkt on page 1401*
- *Speech Parameters Infopkt on page 1402*

End-of-Speech Parameter Infopkt

The end-of-speech parameter infopkt `struct eospkt`, used by infopkt type `INFOPKT_END_OF_SPEECH`, contains the following programmable parameter:

```
struct eospkt
{
   struct infopkt hdr;
   unsigned short mode;
};
```

The following describes this parameter:

Parameter	*mode*	
Units	None.	
Range	0	Stop accepting data from this file, and terminate speech playback.
	1	Stop accepting data from this file, but do not terminate speech playback.
Default	0	

Prompt Map Infopkt

The prompt map infopkt `struct promptpkt`, used by infopkt type `INFOPKT_PROMPT_MAP`, contains the following programmable parameters:

```
struct promptpkt
{
   struct infopkt hdr;
   long num_phrases;
/* map data follows: num_phrases prompt_phrase structures
*/
};
struct prompt_phrase
{
   long phrase_size;
   long offset;
};
```

The following describes each of these parameters:

Parameter	*num_phrases*
Units	None.
Range	1 – `MAX_PHRASES_PER_MAP`.
Description	Number of phrases in prompt file.
Default	None.

Parameter	*phrase_size*
Units	Bytes.
Range	1 – unlimited.
Description	The size of a particular phrase.
Default	None.

Parameter	*offset*
Units	Bytes.
Range	1 – unlimited.
Description	Offset of a particular phrase.
Default	None.

Speech Parameters Infopkt

The speech parameters infopkt `struct SPI`, used by infopkt type `INFOPKT_SPEECH_PARAMETERS`, contains the following programmable parameters:

```
struct SPI
{
   struct infopkt hdr;
   unsigned short sample_rate;
   unsigned char coding_format;
   unsigned char bits_per_sample;
   unsigned char afe_rate;
   unsigned char data_fmt;
};
```

The following describes each of these parameters:

Parameter	*sample_rate*
Units	Samples per second.
Range	0 = 6,000
	1 = 8,000
	6 = 16,000
	2 = 20,000
	3 = 24,000
	4 = 28,000
	5 = 32,000
	9 = 5,300
	10 = 6,300
	11 = 13,000
Default	1

Parameter *coding_format*

Units None.

Range 0 = CVSD
1 = ADPCM
2 = μ - law PCM
7 = G723_1
8 = G729_A
9 = SX7300
10 = SX9600
14 = GSM 610
15 = GSM 660
16 Raw data pass through

Default 1

Parameter *bits_per_sample*

Units Bits per sample.

Range 0 = 1 bit
2 = 4 bits
3 = 8 bits

Default 2

Parameter *afe_rate*

Units Samples per second.

Range 0 = 8,000

Default 0

Parameter *data_fmt*

Units None.

Range 0 = MSB

Default 0xff

Parameter *agc* (not used, for compatibility only)

Units None.

Range A module reported optimal AGC value for playback.

Note: For a list of valid combinations of *sample_rate* and *coding_format*, see *BfvSpeechRecord*.

Fax Infopkt Parameters

Fax infopkt parameters include the following:

- *ASCII Strip Infopkt on page 1404*
- *Document Parameters Infopkt on page 1406*
- *Enhanced Fax Format Page Infopkt on page 1408*
- *Fax Header Parameters Infopkt on page 1409*
- *G3 Strip Infopkt on page 1410*
- *Page Parameters Infopkt on page 1412*
- *T.30 Parameters Infopkt on page 1414*
- *Beginning of Page Infopkt on page 1417*

ASCII Strip Infopkt

The ASCII strip infopkt `struct asciistrippkt`, used by infopkt type `INFOPKT_ASCII_STRIP_PARAMETERS`, contains the following programmable parameters:

```
struct asciistrippkt {
    struct infopkt hdr;
    unsigned short resolution;
    unsigned short width;
    unsigned short eof_char;
    unsigned short font_no;
    unsigned short left_margin;
    unsigned short right_margin;
    unsigned short line_spacing;
};
```

The following pages describe each of these parameters.

Parameter	*resolution*	
Units	None	
Range	0	200H x 100V (Normal)
	1	200H x 200V (Fine)
	2	200H x 400V
	3	300H x 300V
	4	400H x 400V
	5	600H x 600V
	6	1200H x 1200V
	7	300H x 600V
	8	400H x 800V
	9	600H x 1200V
Default	0	

Parameter	*width*	
Units	Pixels	
Range	0	A4, 215 mm, 1728 pixels, normal resolution
	1	B4, 255 mm, 2048 pixels, normal resolution
	2	A3, 303 mm, 2432 pixels, normal resolution
Default	0	

Parameter	*eof_char*
Units	None
Range	ASCII only
Default	1A

Parameter	*font_no*
Units	None
Range	0 – 6 If this specified font has not been downloaded, a default font is used (see *BfvFaxDownloadFont* or the *font_file* parameter on *page 1163*).
Default	0

Parameter	*left_margin*	
Units	1/10 inch	
Range	0	Minimum
	12	Maximum
Default	5	

Parameter	*right_margin*	
Units	1/10 inch	
Range	0	Minimum
	12	Maximum
Default	0	

Parameter	*line_spacing*
Units	Number of G3 lines between text lines.
Range	0 – 255
Default	2

Document Parameters Infopkt

The document parameters infopkt `struct docpkt`, used by infopkt type `INFOPKT_DOCUMENT_PARAMETERS`, contains the following programmable parameters:

```
struct docpkt {
    struct infopkt hdr;
    unsigned char resolution;
    unsigned char hor_width;
    unsigned char vert_length;
};
```

The following page describes each of these parameters.

Parameter	*resolution*	
Units	None	
Range	0	200H x 100V (Normal)
	1	200H x 200V (Fine)
	2	200H x 400V
	3	300H x 300V
	4	400H x 400V
	5	600H x 600V
	6	1200H x 1200V
	7	300H x 600V
	8	400H x 800V
	9	600H x 1200V
	10	100H x 100V (for JPEG only)
Default	0	

Parameter	*hor_width*	
Units	Pixels	
Range	0	A4, 215 mm, 1728 pixels, normal resolution
	1	B4, 255 mm, 2048 pixels, normal resolution
	2	A3, 303 mm, 2432 pixels, normal resolution
Default	0	

Parameter	*vert_length*
Units	None
Range	Reserved, value must be 0
Default	0

Enhanced Fax Format Page Infopkt

The enhanced fax format infopkt `struct effpagepkt`, used by infopkt type `INFOPKT_EFF_PAGE_PARAMETERS`, contains the following programmable parameter:

```
struct effpagepkt {
    struct infopkt hdr;
    unsigned long eff_page_type;
};
```

The following describes this parameter:

Parameter	*eff_page_type*	
Units	None.	
Range	0x1	JPEG
	0x2	Full color (JPEG)
	0x4	Default Huffman Tables (JPEG)
	0x8	12 bits/pel, otherwise 8 (JPEG)
	0x10	No subsampling (JPEG)
	0x20	Custom Illuminant (JPEG)
	0x40	Custom Gamut (JPEG)
	0x0100	JBIG
	0x0200	L0 Mode (JBIG)
Default	0x0	

Fax Header Parameters Infopkt

The fax header parameters infopkt `struct faxhdrpkt`, used by infopkt type `INFOPKT_FAX_HDR`, contains the following programmable parameters:

```
struct faxhdrpkt {
    struct infopkt hdr;
    unsigned char placement;
    unsigned char insert_mode;
};
```

The following describes each of these parameters:

Parameter	*placement*	
Units	None	
Range	0	Header
	1	Footer
Default	0	

Parameter	*insert_mode*	
Units	None	
Range	0x00	Disable
	0x02	Replace
	0x03	Insert
Default	0x03	

ASCII data for the label format follows the fax header parameters infopkt structure (see the *BfvFaxHeader* function).

G3 Strip Infopkt

The G3 strip infopkt `struct g3strippkt`, used by infopkt type `INFOPKT_G3_STRIP_PARAMETERS`, contains the following programmable parameters:

```
struct g3strippkt {
    struct infopkt hdr;
    unsigned short resolution;
    unsigned short width;
    unsigned short data_fmt;
};
```

The following describes each of these parameters:

Parameter	*resolution*	
Units	None	
Range	0	200H x 100V (Normal)
	1	200H x 200V (Fine)
	2	200H x 400V
	3	300H x 300V
	4	400H x 400V
	5	600H x 600V
	6	1200H x 1200V
	7	300H x 600V
	8	400H x 800V
	9	600H x 1200V
Default	0	

Parameter	*width*	
Units	Pixels.	
Range	0	A4, 215 mm, 1728 pixels, normal resolution
	1	B4, 255 mm, 2048 pixels, normal resolution
	2	A3, 303 mm, 2432 pixels, normal resolution
Default	0	

Parameter	*data_fmt*
Units	None
Range	********** MH data format ********** 0x0 EOLs not byte-aligned, MSB. 0x1 EOLs not byte-aligned, LSB. 0x2 EOLs byte-aligned, MSB. 0x3 EOLs byte-aligned, LSB. ********** MR data format ********** 0x4 EOLs not byte-aligned, MSB. 0x5 EOLs not byte-aligned, LSB. 0x6 EOLs byte-aligned, MSB. 0x7 EOLs byte-aligned, LSB. ********** PCX format ********** 0x9 Intel Bi-Level PCX. ********** MMR data format ********** 0x10 EOLs not byte-aligned, MSB. 0x11 EOLs not byte-aligned, LSB. 0x12 EOLs byte-aligned, MSB. 0x13 EOLs byte-aligned, LSB.
Default	0x0 for transmission 0x2 for reception

Page Parameters Infopkt

The page parameters infopkt `struct pageparampkt`, used by infopkt type `INFOPKT_PAGE_PARAMETERS`, contains the following programmable parameters:

```
struct pageparampkt {
    struct infopkt hdr;
    unsigned short top_margin;
    unsigned short bottom_margin;
    unsigned short length;
    unsigned short ascii_pad;
    unsigned short image_pad;     // no longer used
    unsigned short image_break;   // no longer used
    unsigned short image_margin;  // no longer used
};
```

The following describes each of these parameters:

Parameter	*top_margin*
Units	1/10 inch
Range	0 Minimum 25 Maximum
Default	3

Parameter	*bottom_margin*
Units	1/10 inch
Range	0 Minimum 25 Maximum
Default	3

Parameter	*length* (page length) For compatibility only; no longer used.
Units	Number of G3 lines per page in normal resolution (at 98 lines per inch)
Range	0 – 65,535
Default	1,143

Parameter *ascii_pad*

Replaces the *image_pad* parameter and pads all kinds of faxes (not limited only to ASCII images).

Units None.

Range
0 Do not pad short image pages.
1 Pad short image pages.

Default 1 (TR114); 0 (all other modules)

Parameter *image_pad*

For compatibility only; no longer used.

Units None.

Range
0 Do not pad short image pages.
1 Pad short image pages.

Default 0

Parameter *image_break*

For compatibility only; no longer used.

When *image_break* is enabled and the *BfvFaxHeader* function is used, the firmware does not put the fax header data on the second page.

Units None.

Range
0 Do not break long image pages.
1 Break long image pages.

Default 0

Parameter *image_margin*

For compatibility only; no longer used.

Units None.

Range
0 Do not use margins for image pages.
1 Use margins for image pages.

Default 0

T.30 Parameters Infopkt

The T.30 parameters infopkt `struct t30parampkt`, used by infopkt type `INFOPKT_T30_PARAMETERS`, contains the following programmable parameters:

```
struct t30parampkt {
    struct infopkt hdr;
    unsigned char bit_rate;
    unsigned char scan_time;
    unsigned char coding_type;
    unsigned char modulation_type;
    unsigned char line_compression;
    unsigned int iaf_bit_rate;
    char xmt_level;
    char rcv_level;
};
```

The following describes each of these parameters:

Parameter	*bit_rate*	
Units	Bits per second.	
Range	0	2400 V.27, V.34.
	1	7200 V.29, V.17, V.34.
	2	4800 V.27, V.34.
	3	9600 V.29, V.17, V.34.
	4	12000 V.33, V.17, V.34.
	5	14400 V.33, V.17, V.34.
	6	16800, V.34.
	7	19200, V.34.
	8	21600, V.34.
	9	24000, V.34.
	10	26400, V.34.
	11	28800, V.34.
	12	31200, V.34.
	13	33600, V.34.
	0xFF	Any, no restriction

Note: TruFax® does not support values 6 through 13.

Default	0xFF	

Parameter	*scan_time*	
Units	Milliseconds.	
Range	7	0 ms.
	1	5 ms.
	2	10 ms.
	0	20 ms.
	4	40 ms.
Default	7	

Parameter	*coding_type*	
Units	None.	
Range	Reserved, value must be 0.	
Default	0	

Parameter	*modulation_type*	
Units	None.	
Range	0	Any, no restriction.
	1	V.27 only.
	2	V.29 only.
	3	V.33 only.
	4	V.17 only.
	5	V.34 only. TruFax® does not support this value.
Default	0	

Parameter	*line_compression*	
Units	None.	
Range	0	No restriction.
	1	MH only.
	2	MR or MH.
	3	MMR, MR, or MH.
Default	0	

Note: This parameter overrides the user configuration file *line_compression* parameter.

Parameter *iaf_bit_rate*

Units bits per second

Range 0
As fast as possible.

Actual bit rate
For example, 150000 means 150,000 bps.

Default 0

Parameter *xmt_level*

Units 0.5 dB

Range

0	No transmit level adjustment.
1	+0.5 dB
2	+1.0 dB
3	+1.5 dB
.	
.	
-1	-0.5 dB
-2	-1.0 dB
-3	-1.5 dB
.	
.	

Note: Specifies a relative value which is additive to the *btcall.cfg* value xmt_level_override. This xmt_level value will either add or subtract from the transmit level.

Default 0

Parameter *rcv_level*

Units 0.5 dB

Range	0	No receive level adjustment.
	1	+0.5 dB
	2	+1.0 dB
	3	+1.5 dB
	.	
	.	
	-1	-0.5 dB
	-2	-1.0 dB
	-3	-1.5 dB
	.	
	.	
	Note: Specifies a relative value which is additive to the *btcall.cfg* value rcv_level_override. This rcv_level value will either add or subtract from the receive level.	
Default	0	

Beginning of Page Infopkt

The beginning of page infopkt `struct boppkt`, used by infopkt type `INFOPKT_BEGINNING_OF_PAGE`, contains no programmable parameters.

```
struct boppkt {
   struct infopkt hdr;
};
```

This infopkt signals the end of a fax page and the beginning of a new one.

F - Call Progress Notes

This appendix explains the Bfv API's call progress capabilities.

It has the following sections:

- *Processing Call Progress Signals*
- *Adapting to International Specs*
- *Reporting Call Progress Results*
- *Initiating Call Progress*
- *Setting the Call Progress Mode*
- *Special Call Progress Features*
- *Call Progress Signals*
- *Special Information Tones*
- *Custom Call Progress Results*
- *Final Call Progress Results*

Central Offices (COs), telephone carriers, and Private Branch Exchanges (PBXs) generate call progress signals before, during, and after dialing. Dialogic® Brooktrout® boards receive these signals over the telephone lines, and the board's call progress analysis process interprets them.

During call progress analysis, boards can report dial tone detection, ring-back, busy signals, remote fax tone detection, and other important information. Applications can use this information to determine their next course of action, to display the status of a call, or to track billing information. Applications can use postdialing results, such as `HUMAN` and `BUSY`, to decide what redialing strategy to use.

Processing Call Progress Signals

All Dialogic® Brooktrout® boards process call progress signals similarly. Because the frequency characteristics of many of the signals the boards receive are between 300 and 640 Hz, the boards use a bandpass filter to detect energy within that range. The boards use discrete filters to detect other signals with frequency characteristics that fall outside the bandpass filter's range, such as fax answer tone (CED) and Special Information Tones (S.I.T.).

The boards use the bandpass filter and a cadence detection algorithm to detect normal call progress signals (such as, ring-back, busies, and dial tone). When the bandpass filter detects energy in the 300 to 640 Hz range, it generates an ON output. When the bandpass filter fails to detect energy in that range, it generates an OFF output. Over time, the received signals produce a cadence, or pattern of ON/OFF states. The cadence detection algorithm, executing in software and on-board, translates these patterns of ON/OFF states and their duration into call progress results. Because of the cadence algorithm, valuable processing resources on the host computer (PC) remain available for other uses.

The boards use several discrete filters to detect fax answer tone (CED), fax calling tone (CNG), Special Information Tones (S.I.T.), and G2 fax tones. Since some answering fax machines do not transmit the CED tone, the boards also detect fax V.21 signals. V.21 is the modulation that fax machines use to communicate protocol information to each other. So, to ensure that the boards properly detect all G3 fax machines when they dial out, the call progress analysis process must be able to detect V.21.

Although Dialogic® Brooktrout® boards support G3 facsimile only, if, during call progress analysis, the discrete filters detect a fax machine that supports G2 only, call progress analysis reports `G2DETCT`. However, when the boards dial out to fax machines that support both G2 and G3, call progress analysis reports `ANSWER_TONE_DETECT`.

Adapting to International Specs

Brooktrout's call progress analysis process, detects a wide range of call progress signals, both inside and outside the United States.

The call progress analysis process automatically adapts to different signal characteristics that other countries may generate and recognizes them as call progress signals. For applications that run on boards operating outside the United States, this feature eliminates the need to download special parameters to those boards. When dialing to fax machines in other countries, this feature enables call progress analysis to automatically adapt to the proper call progress conditions at the remote end.

Reporting Call Progress Results

Call progress analysis reports two types of call progress results: intermediate and final. Custom call progress values are treated as intermediate call progress results by *BfvLineOriginateCall* or *BfvLineOrigCallDB*.

Intermediate Results

Intermediate call progress results, such as ring-back and remote-off-hook, indicate that a call is progressing. Applications often use intermediate results to display the status of a call or to track billing information. The intermediate call progress results include: `ANSWER`, `RING1`, `RMTOFFHK`, and `SILENCE`.

`RING1` indicates detection of the type of ring-back signal used in the U.S.A.

In digital environments, the boards report `RMTOFFHK` (remote-off-hook) when the A signaling bit goes active. Japanese boards with Polarity Reversal Detection report `RMTOFFHK` when the reverse side answers the call. Applications can use `RMTOFFHK` to determine exactly when the remote end answered the call. Many applications require tracking such information for billing purposes.

Final Results

Final call progress results, such as busy or fax tone detected, indicate that a call has reached a critical point and requires intervention. At that point, applications must stop call progress and initiate the next step, such as terminate the call or send a fax. The final call progress results are listed in *Final Call Progress Results* on *page 1438*. `HUMAN` is a special final result, and detecting it is a complex process (see ***Detecting Human*** below).

For a complete description of the call progress signals and the S.I.T. tones that the boards report, see *Call Progress Signals* on *page 1427* and *Special Information Tones* on *page 1434*.

DISS - Limited Call Progress Mode

Detection of Incoming Signals during Speech (DISS) is a limited call progress mode. DISS is in effect when call progress is performed during speech record or play, or if explicitly requested by the *diss_only* option of *BfvLineCallProgressEnable*.

The final call progress results that are available include:

- CS_BUSY1
- CS_ROBUSY
- CS_DIALTON
- CS_CNG
- CS_CED
- CS_CUSTOM_DIS_FREQ0
- CS_CUSTOM_DIS_CAD0
- No raw call progress data are available

Detecting Human

HUMAN is a final call progress result and means that a human probably answered the call. Because of the complexities involved, detecting HUMAN is not 100% accurate. Call progress analysis reports HUMAN when the results of the analysis do not match any other pattern. Typically, call progress analysis reports HUMAN after the board detects ring-back for a while and then detects a break in the energy pattern.

When call progress is enabled in fax mode and call progress analysis detects HUMAN, to compensate for intermediary intervention at the remote end, such as shared telephone lines and intelligent switching devices, the board does not report HUMAN immediately. For example, because many fax machines share the same phone line with a telephone, a person will often answer the call, realize the call is for the fax machine, and switch the fax machine on.

In fax mode, the board waits until a timeout period expires before reporting HUMAN (assuming it fails to detect another signal after HUMAN). The timeout period is the ced_timeout timeout, and it is programmable through the user configuration file (e.g., *btcall.cfg*). The default for ced_timeout is 40 seconds, but many applications increase this value to as high as 80 seconds.

Initiating Call Progress

The Bfv API provides a high-level function, *BfvLineOriginateCall*, and a low-level function, *BfvLineCallProgressEnable*, to enable call progress analysis. In addition to enabling call progress analysis, the high-level function dials the telephone number. The low-level function simply enables call progress analysis for a particular calling mode.

BfvLineCallProgressEnable

Some applications use the *BfvLineCallProgressEnable* function to enable call progress. This function accepts several input parameters, including the call protocol code and the calling mode (`ORIGINATE` or `ANSWER`) to use.

After calling this function, applications access the call progress buffer, a 128-byte structure that contains up to sixty-four call progress samples. Each 2-byte sample consists of a code and its associated data.

The call protocol code sets the call progress mode, and the calling mode defines which call progress signals the board expects to receive.

Applications enable the `ANSWER` call mode when responding to an incoming call. For this purpose, applications check the contents of the call progress buffer in a loop, looking for `CS_CNGDTCT` to determine if the call is from a fax machine. When it is in `ANSWER` call mode, the board is also in voice protocol mode and does not transmit CNG tone.

Applications enable the `ORIGINATE` call mode when out dialing. For this purpose, applications check the contents of the call progress buffer in a loop, looking for call progress results to determine what action to take next. When it is in `ORIGINATE` call mode and in fax call protocol mode, the board transmits CNG tone.

The returned data values for call progress results that the board determined, corresponds to the names of the call progress signals described in *Call Progress Signals* on *page 1427* and in *Special Information Tones* on *page 1434*, but with `CS_` appended to the front of the name (for example, `CS_BUSY1`). The *BfvDataCP* function returns these values, and they are defined in the *btlib.h* file.

BfvLineOriginateCall

Most applications use the high-level function *BfvLineOriginateCall* to initiate call progress. This function accepts several input parameters, including the phone number to dial, the call protocol code to use, and a user-defined function.

The *BfvLineOriginateCall* function commands the channel to dial the specified telephone number. Besides the dialing digits, the telephone number may include control parameters, such as “w” (wait for dial tone for the analog boards) and “p” (pulse dial). After dialing the dial string, the function commands the board to perform call progress analysis.

Applications often pass a user-defined function to *BfvLineOriginateCall* to interpret intermediate call progress results (such as, `RING1` and `RMTOFFHK`).

The *BfvLineOriginateCall* function returns when one of the following occurs:

- An error occurs during dialing.
- The channel reports a final call progress result.
- The user-defined function aborts the call.

When *BfvLineOriginateCall* returns, the application can query the `call_result` structure to determine what happened.

Setting the Call Progress Mode

In general, the `call_protocol_code` sets three call progress modes – `VOICE`, `FAX`, and `RAW` – that affect how the boards perform call progress analysis and report the call progress results. Applications use each of these modes for specific purposes.

Voice Mode

Applications enable this mode when dialing to a human for voice applications, such as voice notification or automated soliciting. For this purpose, we recommend that applications pass *BfvLineOriginateCall* the `CALL_PROTOCOL_VOICE_NO_RAW` protocol code. In this mode, the board reports `HUMAN` and other answer results immediately.

Fax Mode

Applications enable this mode when dialing to a fax machine. For this purpose, You should ensure that applications pass *BfvLineOriginateCall*, the `CALL_PROTOCOL_FAX` protocol code. In this mode, the module suppresses `HUMAN` and `QUIET` results until the `ced_timeout` timeout expires.

Note: `CALL_PROTOCOL_FAX_NO_RAW` (fax mode) and `CALL_PROTOCOL_VOICE_NO_RAW` (voice mode) report only the results of the module's call progress analysis, not the raw call progress data.

Raw Mode

Applications enable this mode to receive the raw energy levels that the call progress filters detect directly from the board. Applications pass *BfvLineOriginateCall*, the `CALL_PROTOCOL_FAX_NO_RAW` protocols code and use this information to construct their own call progress algorithm. However, since the vast majority of applications are not interested in this information, we recommend that they enable one of the other modes to perform the required operation.

Applications can enable raw mode and voice mode or fax mode at the same time, but fax mode and voice mode are mutually exclusive.

Special Call Progress Features

Dialogic® Brooktrout® boards can perform very useful tasks during call progress analysis.

Sending CNG

In fax mode, Dialogic® Brooktrout® boards transmit CNG tone (fax calling tone) while performing call progress analysis. Because this feature guarantees that the remote end will detect a CNG tone when it answers the call, this feature eliminates incompatibility problems with intelligent switching devices. When these devices go off-hook, they listen for CNG tone before deciding whether to connect the call to a fax machine or to a human.

Call Progress Analysis During Dialing

Using call progress analysis when dialing is very useful when the dial string includes the “W” control character.

The “W” control character directs the channel to wait for a dial tone before dialing the next digit in the dial string. Since dial tone is a continuous signal in many countries, to qualify a dial tone, call progress analysis simply looks for continuous energy output from the board’s bandpass filter.

Call Progress Signals

FCP_ANSWER	Remote end answered call; can occur immediately after a break in the ring-back cycle; like `HUMAN`, does not match any other call progress signal patterns, but is marked by silence.
Mode	`VOICE, ORIGINATE`
Result Type	Intermediate or final
Detection	Cadence algorithm
Freq. (Hz)	NA
On/Off secs	NA
Action	Voice applications only; stop call progress immediately and play speech or wait until `HUMAN` (and possibly `SILENCE`) detected and play speech.

FCP_ANSWER_TONE_DETECT

	Fax machine detected; usually a fax CED tone, but also fax V.21 signals when the remote machine does not send a CED tone before it sends the fax protocol information.
Mode	`FAX, VOICE, ORIGINATE, ANSWER`
Result Type	Final
Detection	Discrete filters
Freq. (Hz)	`2100` or `V.21` (1650, 1850)
On/Off secs	Continuous for >2.6 and <4 or V.21 (NIA)
Action	Send a fax.

FCP_BUSY1	Normal busy; remote end busy (off-hook).
Mode	`FAX, VOICE, ORIGINATE`
Result Type	Final
Detection	Cadence algorithm
Freq (Hz)	Country-specific (`480+620` U.S.A.)
On/Off secs	Country-specific (`0.5/0.5` U.S.A.)
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. Employ suitable redial strategy; for example, redial every 10 minutes for several hours.

FCP_BUSY2	Normal busy; remote end busy (off-hook). Used instead of `BUSY1` in certain countries.
Mode	`FAX, VOICE, ORIGINATE`
Result Type	Final
Detection	Cadence algorithm
Freq (Hz)	Country-specific
On/Off secs	Country-specific
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. Employ suitable redial strategy; for example, redial every 10 minutes for several hours.
FCP_CNGDETCT	CNG fax calling tone detected.
Mode	`ANSWER`; typically, applications answer inbound calls, go off-hook, check for CNG, and depending on the results, determine whether to receive a fax or play speech. (Assuming the remote end started sending CNG when call progress was initiated, worst case detection time is 5 secs.)
Result Type	Final
Detection	Discrete filters
Freq (Hz)	1100
On/Off secs	0.5/3.0
Action	If detected, terminate call progress with the *BfvLineCallProgressDisable* function or the user-defined function passed to the *BfvLineOriginateCall* function, and start fax receive mode.
FCP_CONFIRM	Confirmation tone; automated equipment acknowledges successful completion of caller requested feature (for example, call forwarding). This is not G2 confirmation tone (CFR2).
Mode	`FAX, VOICE, ORIGINATE`
Result Type	Intermediate
Detection	Cadence algorithm
Freq (Hz)	350+440
On/Off secs	`2((0.1/0.1),0.1/continuous`
Action	When the application receives this result, it should continue call progress analysis and use the user-defined function passed to *BfvLineOriginateCall* to check for `CONFIRM` and display the status of the call.

FCP_DIALTON	Dial tone detected; usually indicates the dialing sequence did not break dial tone.
Mode	`FAX, VOICE, ORIGINATE`
Result Type	Final
Detection	Cadence algorithm
Freq (Hz)	350+440
On/Off secs	Continuous
Action	Terminate the connection with *BfvLineTerminateCall* or *BfvLineReset*. Retry 10 minutes later. If still unsuccessful, have the telephone company check the line, and then check the Dialogic® Brooktrout® board (if the volume of the DTMF tones is too low, the PBX or the telephone company will fail to detect them).
FCP_G2DETCT	Group 2 fax machine detected; remote machine is capable of sending and receiving G2 facsimiles only.
Mode	`FAX`, `VOICE`, `ORIGINATE`
Result Type	Final
Detection	Discrete filters
Freq (Hz)	1850
On/Off secs	1.5/3.0
Action	Since the Bfv API does not support G2, terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. Retry only once, 10 to 15 minutes later. If still unsuccessful, check the telephone number.
FCP_HUMAN	Answer (probable human) detected; does not match any other expected call progress signal patterns.
Mode	`FAX` (suppressed until after `ced_timeout` timeout), `VOICE, ORIGINATE, ANSWER`
Result Type	Final
Detection	Cadence algorithm
Freq (Hz)	NA
On/Off secs	NA
Action	Application specific. Fax applications: terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. Employ a suitable retry strategy; for example, redial once 30 to 60 minutes later. Voice applications: play a speech file.

FCP_ISDN_CALL_PROGRESS

By enabling call progress on an ISDN D channel, one of the following values will be in the second byte of the FIFO buffer:

4: `CALL_PROCEEDING`:	Call is proceeding normally.
5: `CALL_ALERTING`:	Ringback detected; remote end is ringing.
6: `CALL_CONNECTED`:	Call is connected.
7: `CALL_DISCONNECTED`:	Call was disconnected.

Mode	ORIGINATE
Result Type	Intermediate
Detection	NA
Freq (Hz)	NA
On/Off secs	NA
Action	Depending on the call progress value, proceed as in fax or voice applications.

FCP_ISDN_CALL_COLLISION

Indicates that a call collision occurred on the ISDN line.

Mode	Originate
Result Type	Final
Detection	NA
Freq (Hz)	NA
On/Off secs	NA
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*.

FCP_PULSE

This result is reserved and should never occur.

Mode	None.
Result Type	NA
Detection	NA
Freq (Hz)	NA
On/Off secs	NA
Action	NA

FCP_QUIET	After dialing the number, no energy detected on the line for the `ced_timeout` timeout period; possible dead line.
Mode	`FAX, ORIGINATE`
Result Type	Final
Detection	Cadence algorithm and discrete filters
Freq (Hz)	None
On/Off secs	None
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. Redial every 10 minutes, up to 2 hours. This result is atypical, so check the telephone number if redialing is unsuccessful. Check any channel that reports this result excessively.
FCP_RECALL	Recall dial tone detected; signal generated when calling another party while already connected to one or more parties (for example, conference calling, call waiting).
Mode	`FAX, VOICE, ORIGINATE`
Result Type	Final
Detection	Cadence algorithm
Freq (Hz)	350+440
On/Off secs	3+(0.1/0.1), Continuous
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. This result is atypical, and unless specifically expected, it may be an invalid result. If detected after dialing to a fax machine, redial the number 10 minutes later.
FCP_RING1	Ringback detected; remote end is ringing. The Central Office connected to the dialed number generates this signal.
Mode	`FAX, VOICE, ORIGINATE`
Detection	Cadence algorithm
Result Type	Intermediate
Freq (Hz)	440+480
On/Off secs	`2/4` or `1/3` (PBX extensions)
Action	Continue call progress analysis. Can use the user-defined function passed to *BfvLineOriginateCall* to check for ringback and display the status of a call.

FCP_RMTOFFHK

Remote fax machine went off-hook (also known as Answer Supervision). Depending on the configuration of the CO, T1 and E1 connections may not use or provide in-band signaling.

Mode	`FAX, VOICE, ORIGINATE`
Result Type	Intermediate
Detection	MVIP A signaling bit
Action	Continue call progress analysis. Can pass a user function to *BfvLineOriginateCall* to check for this result and use it to track call status and billing information.

FCP_RNGNOANS

Indicates the remote end was ringing but did not answer. In fax mode, this result occurs after the `wait-for-ced` timeout has expired and the line continues to ring. In voice mode, this result occurs after the `v_timeout` has expired and the line continues to ring. (You can adjust the value of these timeout parameters in the user configuration file, *btcall.cfg*.

Mode	`FAX, VOICE, ORIGINATE`
Result Type	Final
Detection	Cadence algorithm
Freq (Hz)	440+480
On/Off secs	See *FCP_RING1* on *page 1431* .
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. The retry strategy depends on the application. For example, redial the number every 15 minutes for 2 hours. After that, redial only once every hour or two.

FCP_ROBUSY

Reorder or fast busy; indicates that telephone company trunk lines are busy; on PBXs, indicates no available outside lines.

Mode	`FAX, VOICE, ORIGINATE`
Result Type	Final
Detection	Cadence algorithm
Freq (Hz)	Country-specific (480+620 U.S.A.)
On/Off secs	Country-specific (0.25/0.25 U.S.A.)
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. Redial the number every 10 minutes for 2 hours. After that, check the number.

FCP_SILENCE	In `VOICE` mode, after dialing, no signal detected during the silence timeout. In `ANSWER` mode, no fax CNG tone detected after answering a call.
Mode	`VOICE, ORIGINATE, ANSWER`
Result Type	Intermediate (usually)
Detection	Bandpass filter (no energy detected)
Freq (Hz)	None
On/Off secs	None
Action	Application specific. In `ORIGINATE` mode, use this result with the user-defined function passed to *BfvLineOriginateCall* to determine when a human on the remote end has stopped speaking; then abort call progress and play a speech file. In `ANSWER` mode, use this result to determine when the call is from a human; then stop call progress analysis immediately and play a speech file.
FCP_SPECIALCP	Special call progress tone detected.
Mode	FAX, VOICE, ORIGINATE
Result Type	Intermediate
Detection	Discrete filters
Freq (Hz)	950/1400
On/Off secs	Greater than 0.1
Action	Continue call progress analysis.

Special Information Tones

Special Information Tones (S.I.T.) are useful in determining where and why an attempted call failed. This special information includes customer originating failures, intercept and vacant; end office originating failures, no circuit-BOC and reorder-BOC; and carrier failures, no circuit-CXR and reorder-CXR.

Note: Once it detects an S.I.T. tone, the board uses tone duration patterns, not tone frequencies, to identify the particular S.I.T. tone. So, the board does not distinguish between no circuit and reorder failures originating from an end office and those originating from the carrier.

FCP_SITINTC

Intercept tone detected; remote end originating failure; invalid telephone number or class of service restriction.

Mode	`FAX, VOICE, ORIGINATE`
Result Type	Final
Detection	Discrete filters
Freq (Hz)	913.8/1370.6/1776.7
On/Off secs	0.274/, 0.274/, 0.380/
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. Redial 10 to 15 minutes later. If still unsuccessful, check the number.

FCP_SITNOCIR

No circuit detected; end office or carrier originating failure, possible dead line.

Mode	`FAX, VOICE, ORIGINATE`
Result Type	Final
Detection	Discrete filters
Freq (Hz)	985.2(913.8)/1428.5(1370.6)/1776.7
On/Off secs	0.380/, 0.380/, 0.380/
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. Redial every 10 to 15 minutes for 2 hours. If still unsuccessful, check the telephone number. If correct, report it to the telephone company.

FCP_SITREORD	Reorder tone detected; end office (PBX) or carrier originating failure.
Mode	`FAX, VOICE, ORIGINATE`
Result Type	Final
Detection	Discrete filters
Freq (Hz)	985.2(913.8)/1428.5(1370.6)/1776.7
On/Off secs	0.274/, 0.380/, 0.380/
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. Redial every 10 to 15 minutes for 2 hours. If still unsuccessful, check the telephone number. If correct, report it to the telephone company.
FCP_SITVACODE	Vacant tone detected; remote originating failure; invalid telephone number.
Mode	`FAX, VOICE, ORIGINATE`
Result Type	Final
Detection	Discrete filters
Freq (Hz)	985.2/1370.6/1776.7
On/Off secs	0.380/, 0.274/, 0.380/
Action	Terminate the call with *BfvLineTerminateCall* or *BfvLineReset*. Redial 10 to 15 minutes later. If still unsuccessful, check the telephone number.

Custom Call Progress Results

The Brooktrout fax boards support custom programmable call progress detection. Call progress results may be programmed based on either frequencies or cadence. See *BfvLineCallProgressProgram* in *Volume 3*.

When programming a custom call progress value for standard call progress mode, one of eight templates may be chosen for frequency and one of three may be chosen for cadence. In limited call progress mode (DISS), only one template may be chosen which may be either frequency or cadence. The possible call progress results shown below reflect the template value chosen.

CS_CUSTOM_FREQ0
CS_CUSTOM_FREQ1
CS_CUSTOM_FREQ2
CS_CUSTOM_FREQ3
CS_CUSTOM_FREQ4
CS_CUSTOM_FREQ5
CS_CUSTOM_FREQ6
CS_CUSTOM_FREQ7

Custom programmed call progress frequency template N (where N= 0-7) was detected.

Mode	All
Result Type	Intermediate
Detection	Cadence algorithm and discrete filters
Frequency	300 – 3200 Hz
Duration	10 ms units
Action	Continue call progress analysis unless application terminates it.

CS_CUSTOM_CAD0
CS_CUSTOM_CAD1
CS_CUSTOM_CAD2
CS_CUSTOM_CAD3
CS_CUSTOM_CAD4
CS_CUSTOM_CAD5
CS_CUSTOM_CAD6
CS_CUSTOM_CAD7

	Custom programmed call progress cadence template N (where N= 0-7) was detected.
Mode	All
Result Type	Intermediate
Detection	Cadence algorithm and discrete filters — 300 – 600 Hz
On/Off secs	10 ms units
Action	Continue call progress analysis unless application terminates it.

CS_CUSTOM_DIS_FREQ0

	Custom programmed call progress frequency template 0 for limited mode (DISS) was detected.
Mode	All
Result Type	Intermediate
Detection	Cadence algorithm and discrete filters
Frequency	300 – 3200 Hz
Duration	10 ms units
Action	Continue call progress analysis unless application terminates it.

CS_CUSTOM_DIS_CAD0

	Custom programmed call progress cadence template 0 for limited mode (DISS) was detected.
Mode	All
Result Type	Intermediate
Detection	Cadence algorithm and discrete filters — 300 – 600 Hz
On/Off secs	10 ms secs
Action	Continue call progress analysis unless application terminates it.

Final Call Progress Results

FCP_BUSY1	301
FCP_BUSY2	302
FCP_ROBUSY	303
FCP_RECALL	304
FCP_CONFIRM	305
FCP_PULSE	306
FCP_HUMAN	316
FCP_ANSWER	317
FCP_DIALTON	318
FCP_SILENCE	324
FCP_RNGNOANS	325
FCP_G2DETCT	326
FCP_SITINTC	327
FCP_QUIET	328
FCP_SITVACODE	329
FCP_SITREORD	330
FCP_SITNOCIR	331
FCP_CNGDETCT	332
FCP_ANSWER_TONE_DETECT	339
FCP_UNKNOWN	340
FCP_ISDN_CALL_PROGRESS	348
FCP_ISDN_CALL_COLLISION	349

G - Country-Specific Parameter Files

This appendix describes the country-specific parameter files and dialing requirements.

It has the following sections:

- *BT_CPARM.CFG Parameter File*
- *Using Dialing Database Functions and Dialing Parameters*
- *Country-Specific Dialing Requirements*
- *Examples of R2 Parameter Files*

Many countries have dialing requirements that regulate how telephony equipment must interface with the telephone network. If an application dials a fax machine in one of these countries, it must use the appropriate `country_code` keyword in the user-defined configuration file and the dialing database functions. These tools, used in conjunction with the dialing database configuration parameters in the *BT_CPARM.CFG* configuration file, enable applications to conform to the target country's PTT regulations.

The following countries have approved one or more Dialogic® Brooktrout® TR Series boards:

- Australia
- Canada
- EU (TBR 4)
- Japan
- United States

The section, *Country-Specific Dialing Requirements on page 1450* describes the unique dialing requirements for individual countries.

For information about R2 parameter files for Argentina, Brazil, China, Korea and Mexico, see the examples on *page 1459*.

The application is responsible for enforcing the dialing restrictions of any country that requires it. For a description of the dialing restriction parameters in the *BT_CPARM.CFG* file, see *BT_CPARM.CFG Parameter File*. For specific details about the dialing database functions, see *Volume 2, Chapter 4, Dialing Database Functions*.

BT_CPARM.CFG Parameter File

The Brooktrout SDK includes a read-only file called the *BT_CPARM.CFG* parameter file. This file contains several sets of parameters that primarily relate to telephony configuration (for example: DTMF tone length, interdigit times). A set of parameters is specific to a country or to a group of countries. Since some PTTs require specific values for these parameters, Dialogic provides the values in this file. Dialogic also guarantees that parameter files created for subsequent Bfv API versions will be compatible with previous versions.

The *BT_CPARM.CFG* file resides in the *app.src* and *bapp.src* directories. You can create it with the *mkparams* program, included in source form in the *app.src/params* subdirectory. Dialogic strongly recommends that you use the supplied values.

The location of *BT_CPARM.CFG* must be specified in the user-defined configuration file (see the *bt_cparm* parameter on *page 1152.* The default value is *BT_CPARM.CFG* in the current directory).

All parameters are declared as unsigned chars unless otherwise noted, and are as follows:

Parameter	Value
blind_dial	Disables initial dial tone detection when any nonzero value is indicated. Dialing begins after this amount of time has expired and after going off-hook. 1 second units
data_level	Indicates the data transmit level. 0.5 dBm units
dial_tone_min	Indicates the minimum time to identify the presence of a dial tone as a dial tone during the initial “w” wait for dial tone command. 10 ms units
dtmf_high_level	Indicates the DTMF upper frequency transmit level. 0.5 dBm units
dtmf_low_level	Indicates the DTMF lower frequency transmit level. 0.5 dBm units

Parameter	Value
dtone_len	Indicates the minimum time to identify the presence of a dial tone as a dial tone during a wait for dial tone command. The time refers to the 2nd, 3rd, 4th, ... “w” in the dial string. To allow the call progress algorithm to work properly, this parameter is set to a value that exceeds the longest expected ring-back “ON” period. 50 ms units
dtone_timeout_highbyte	Indicates the timeout period to wait for a dial tone (high byte). 100 ms units
dtone_timeout_lowbyte	Indicates the timeout period to wait for a dial tone (low byte). If the system fails to detect a dial tone within the time period specified by these high and low byte parameters, the system considers the line dead and returns the corresponding indication to the application. 100 ms units
enable_loop_cur	Enables/disables loop current monitoring. 1 Enabled 0 Disabled
fixed_ced	Defines a fixed or variable `ced_timeout` timeout. If variable, the timeout can be changed via the `ced_timeout` (see *page 1154*) user parameter. 1 Fixed 0 Variable
fixed_id	Defines a fixed or variable local ID string. If fixed, the Bfv API uses the value specified in the *id_string* (see *page 1164*) parameter in the user-defined configuration file. If variable, the Bfv API still uses the *id_string* parameter but this value can be overridden by a call to the *BfvFaxSetLocalId* function. 1 Cannot change local ID for the CSI, TSI, and CIG CCITT frames. 0 Permit variable local ID strings.
loopcur_debounce	Indicates the time allowed for interruptions in loop current. The system ignores any interruptions in loop current that occur for less than this limit. 50 ms units Maximum 1.250 seconds Default 14 hex (1 second)

Parameter	Value
loopcur_len	Indicates the minimum amount of time that loop current must be present to indicate the presence of an incoming call. 50 ms units Default 2 (100 ms)
loopcur_timeout	Indicates the timeout period to wait for loop current. If the system fails to detect loop current within the specified time period, the system considers the line dead and returns the corresponding indication to the application. 50 ms units
loop_max_break	Indicates the maximum break time for loop curent. If the Bfv API does not detect loop current for this period of time during a call, it assumes the call disconnected. 5 ms units
loop_seizure	Indicates the amount of loop seizure time required to indicate a valid incoming call. 5 ms units
max_bitrate	Indicates the maximum transmission bit rate. Valid values are (bps units): 0 2400 2 4800 1 7200 3 9600 4 12000 5 14400 6 16800 7 19200 8 21600 9 24000 10 26400 11 28800 12 31200 13 33600 0xFF generic; maximum supported by hardware/firmware

Parameter	Value
max_interdigit	Indicates the maximum time between incoming DID digits. 50 ms units
max_sil_timeout	Indicates the maximum silence timeout when recording speech. When this value is nonzero, the Bfv API uses the shorter silence timeout period: either the value from this parameter or the value from the recording function. The range is 0–255. 100 ms units
min_on_hook	Indicates the minimum length of time to maintain an on-hook state. 50 ms units Default 28 hex (2 seconds)
post_wink	Indicates the minimum length of time that must elapse at the end of the wink signal before the firmware generates a call detected interrupt, even if the system has detected all the expected incoming digits. In wink mode, the timer starts at the end of the wink (before the 40-ms debounce begins). In immediate mode, the timer starts after detecting the end of a valid loop seizure that indicates receipt of a valid incoming call. 50 ms units
pre-wink	Indicates the minimum length of time the system must wait (pause) after detecting the end of a valid loop seizure (incoming call) before it sends a wink signal. 5 ms units
pulse_break	Indicates the break time for pulse dialing. 5 ms units
pulse_inter_time	Indicates the time between digits during pulse dialing. 5 ms units
pulse_make	Indicates the make time for pulse dialing. 5 ms units
pulse_max_break	Indicates the maximum break time for pulse dialing. When the system detects a break time between pulse-dialed digits that exceeds the value in this parameter, the system considers the digit invalid. 5 ms units

Parameter	Value
pulse_min_break	Indicates the minimum break time for pulse dialing. If the system fails to detect a pulse-dialed break time of this minimum duration, the system considers the digit invalid. The system uses the parameter for debouncing digits. 5 ms units
rec_level	Data receive level. 0.5 dBm units
ring_blank	Indicates ringing blank time. This parameter indicates the length of time to wait after detecting a burst of ringing before looking for another burst of ringing. 50 ms units
ring_len	Indicates the minimum length of the ring signal. This parameter indicates a value that the system uses to determine the frequency of the ring signal. 5 ms units
scan_time	Indicates the minimum scan time. Valid values: 7 0 ms 1 5 ms 2 10 ms 0 20 ms 4 40 ms
tone_inter_time	Indicates the amount of silence allowed between DTMF tones during dialing. 5 ms units
tone_len	Indicates the length of a DTMF tone. This parameter defines the basic unit used when playing a DTMF tone. 5 ms units
wait_for_ced_high	Indicates the length of time to wait for the called station's ID signal (high byte and low byte).
wait_for_ced_low	10 ms units

All of the remaining parameters provide control for the dialing restrictions specific to each country the Bfv API supports.

Parameter	Value
dl_configuration	Contains bits that specify the properties of the redial strategy for the country: b0 (1) Restrict dialing based on the telephone number. b1 (2) Restrict dialing based on the telephone line (channel). If neither bit is set, the country has no dialing restrictions. If both bits are set, the restrictions apply only to the telephone numbers, and only minimum intercall delays will affect the telephone lines. b2 (4): 1 The *dl_uns_lmt_time* or *dl_wr_lmt_time* applies after the call that exceeded the limit. Otherwise, the time applies after the first call. b3 (8): 1 For phone line restrictions, do not treat wrong calls as successful. Otherwise, for phone line restrictions, treat wrong calls as successful.
dl_max_limit	Indicates the maximum number of wrong and unsuccessful dial attempts permitted for any telephone number. The Bfv API blacklists a telephone number when this maximum is reached. This value applies only when two maximum retry values, one on a per time basis and one on a forever (blacklist) basis, exist. In this case, the Bfv API assumes that only “sum limits” are in effect. The Bfv API uses *dl_max_limit* to control the forever (blacklist) limit and the *dl_sum_limit* with both the *dl_wr_lmt_time* and the *dl_uns_lmt_time* to control the per time limit. When only one maximum retry value exists, it is either *dl_sum_limit*, *dl_unsucc_limit*, or *dl_wrong_limit*.
dl_min_unsucc	Indicates the minimum time to delay redialing a telephone number after an unsuccessful dial attempt. This delay applies to all redial attempts after the first one. 1 second units
dl_min_unsucc_first	Indicates the minimum time to delay redialing a telephone number after an unsuccessful dial attempt. This delay only applies to the first redial attempt. 1 second units
dl_min_wrong	Indicates the minimum time to delay redialing a telephone number after a wrong dial attempt. This delay applies to all redial attempts after the first one. 1 second units

Parameter	Value
dl_min_wrong_first	Indicates the minimum time to delay redialing a telephone number after a wrong dial attempt. This delay only applies to the first redial attempt. 1 second units
dl_sum_limit	Indicates the maximum number of the sum of unsuccessful and wrong dial attempts permitted. When this number is reached, the Bfv API either blacklists the telephone number or delays redialing it until *dl_wr_lmt_time* or *dl_uns_lmt_time*. 0 Disable.
dl_uns_lmt_time	Indicates the time that applies to the *_unsucc_* or *_sum_* limit. 0 Forever (blacklisted). Nonzero Reset the unsuccessful count to 0 after the time limit elapses. If both *dl_wr_lmt_time* and *dl_uns_lmt_time* are nonzero, they must be equal. Bit b2 of *dl_configuration* affects the application of this parameter. 1 second units
dl_unsucc_limit	Indicates the maximum number of unsuccessful calls permitted. When this number is reached, the Bfv API blacklists the telephone number or delays redialing it until *dl_uns_lmt_time*. 0 Disable
dl_wr_lmt_time	Indicates the time that applies to the *_wrong_* or *_sum_* limit. 0 Forever (blacklisted). Nonzero Reset the wrong count to 0 after the time limit elapses. If both *dl_wr_lmt_time* and *dl_uns_lmt_time* are nonzero, they must be equal. Bits b2 and b3 of *dl_configuration* affect the application of this parameter. 1 second units
dl_wrong_limit	Indicates the maximum number of wrong calls permitted. When this number is reached, the Bfv API blacklists the telephone number or delays redialing it until *dl_wr_lmt_time*. 0 Disable.

Using Dialing Database Functions and Dialing Parameters

The dialing database functions are *BfvDialDBCheck*, *BfvDialDBUpdate*, *BfvLineOrigCallDB*, and *BfvDialDBList*. To properly enforce the restrictions, the application must use either the *BfvLineOrigCallDB* function or the *BfvDialDBCheck* and *BfvDialDBUpdate* functions. See *Volume 2, Chapter 4, Dialing Database Functions* for specific details.

An application uses the dialing database functions only when sending a fax, when it expects a fax machine to answer. An application must not use the dialing database functions when it expects a human or a voice answering machine to answer.

An application uses the dialing database locally on a single computer only. It must not share the dialing database with multiple computers over a network. The Bfv API does not provide for sharing or for format differences in intercompiler time storage.

The dialing database functions use a lock file to ensure exclusive access to the database. The name of this file is *btdb.lck*, and it resides in the same directory as the dialing database. If execution of the program stops prematurely (for example, the system crashes) the lock file may remain. If this occurs, you must manually remove the lock file, so the dialing database functions can proceed.

The Bfv API uses the C library *time()* function for timing purposes. Some libraries implement this function slightly differently. Users must make sure that all programs using the *DialDB*... functions on a particular computer are linked with the same version of their compiler library.

The Bfv API can add trunk access information to the database at the same time as the telephone number or instead of the telephone number. For viewing and removal purposes, the database stores a trunk line under the telephone number `xN`, where `N` is the channel number.

The Bfv API implements the per-time-period restrictions so that a period normally starts with the first failed call. The next call after the period has ended, begins a new period. (So, even with a restriction of five calls per hour, it's possible to have nine calls concentrated in a

very small time span.) Bit `b2` in `dl_configuration` permits the period to start after the last call. No provision is made for a “sliding window”, which records the last `N` call times.

When restrictions are based on the telephone line, *wrong* calls may or may not cause delays, as determined by bit `b3` in `dl_configuration`.

If a country requires a minimum delay between calls that varies depending on how many calls have been made, the Bfv API provides one value for the delay that follows the first call and another value for the delay that follows all subsequent calls. If a country requires a different setup, the Bfv API uses the same method, but with more conservative delay times. The Bfv API predefines a blacklist that contains several telephone numbers that are restricted in certain countries. The application must never dial these telephone numbers within the specified country. These telephone numbers are not specified through the *BT_CPARM.CFG* configuration file, but instead, are hard-coded within the Bfv API source. They include:

Japan 110, 119

Hong Kong 999

Country-Specific Dialing Requirements

There are dialing requirements specific to individual countries. Country requirements which are defined, include the following:

Note: * Indicates countries not yet approved for Dialogic® Brooktrout® boards.

- *Australia on page 1450*
- *Canada on page 1451*
- *Czech Republic on page 1451**
- *Denmark on page 1452*
- *European Community (Boards Approved to TBR 4) on page 1452*
- *France on page 1452*
- *Germany on page 1453*
- *Hong Kong on page 1453**
- *Ireland on page 1453*
- *Israel on page 1454**
- *Italy on page 1454*
- *Japan on page 1454*
- *Malaysia on page 1455**
- *Netherlands on page 1455*
- *New Zealand on page 1455**
- *Norway on page 1456**
- *Singapore on page 1456**
- *Spain on page 1456*
- *Switzerland on page 1457**
- *Turkey on page 1457**
- *United Kingdom on page 1457*
- *United States on page 1458*

Australia

- Regulatory authorities recommend that the application delay two seconds before answering an incoming call.

- If the application does not wait for dial tone, it must wait at least two seconds after seizing the line before dialing. Include `missing_wait 40` in your *callctrl.cfg* file to specify the two-second wait.

Canada

Unsuccessful calls are either unanswered calls or wrong calls. Unanswered calls occur when the called number fails to answer. In this case, the Bfv API reports `BUSY1`, `BUSY2`, `ROBUSY`, `RECALL`, `DIALTON`, `SIT_`, or `RNGNOANS`. Wrong calls occur when the called number answers but fails to send fax CED tone or V.21 signal to indicate a fax machine. In this case the Bfv API reports `HUMAN`, `QUIET`, `SILENCE`, `ANSWER`, or `G2DETCT`.

For transmission of any one document to any one telephone number:

- For unanswered calls, the application must:
  - Make no more than fifteen call attempts in one hour (unless the retry detector is manually reset).
  - Release the line within fifteen seconds after detection of one of the above call progress signals.
- For wrong calls, the application must:
  - Make no more than two call attempts in one hour (unless the retry detector is manually reset).
  - Release the line within fifteen seconds after detection of one of the above call progress signals.

Czech Republic

For transmissions of any one document to any one telephone number, the application must:

- Make no more than twelve call attempts.
- Delay five seconds between the first and second call attempts and between the second and third call attempts.
- Delay one minute between each subsequent call attempt.

For calls to different telephone numbers over the same trunk, the application must delay five seconds between releasing the line after an unsuccessful call attempt and seizing it for the next call attempt.

Denmark

For transmission of any one document to any one telephone number, the application must:

- Make no more than fifteen call attempts.
- Delay five seconds between successive call attempts to the same number.

European Community (Boards Approved to TBR 4)

Pulse dialing is not supported in the European Community (EC) for TBR 4 boards.

For transmission of any one document to any one telephone number, the application must:

- Make no more than fifteen call attempts.
- Delay five seconds between successive call attempts to the same number.

France

In France, unsuccessful calls are categorized as one of two types: inefficient calls or wrong calls. Inefficient calls occur when the called number fails to answer. In this case, the Bfv API reports `BUSY1`, `BUSY2`, `ROBUSY`, `RECALL`, `DIALTON`, `SIT_`, or `RNGNOANS`. Wrong calls occur when the called number answers but fails to send fax CED tone or V.21 signal to indicate a fax machine. In this case the Bfv API reports `HUMAN`, `QUIET`, `SILENCE`, or `ANSWER`.

- For applications that differentiate between inefficient calls and wrong calls, the application must:
  - Make no more than six call attempts per hour.
  - Delay from one to twelve minutes between each call attempt.
  - Add the telephone number to the blacklist and make no more attempts to send any document to that telephone number when it detects a wrong call twice during the hour.

- For applications that do not differentiate between inefficient calls and wrong calls, the application must:
  - Make no more than six call attempts per hour.
  - Delay from one to twelve minutes between each call attempt.
  - Add the telephone number to the blacklist and make no more attempts to send any document to that telephone number after six failed attempts to detect an answering fax machine.
- Only an operator issuing the command manually can remove a telephone number from the blacklist.

Germany

For calls to the same or different telephone numbers over the same trunk, the application must do either of the following:

- Delay thirty seconds between releasing the line after an unsuccessful call attempt and seizing it for the next call attempt.
- Delay five seconds between each call attempt and two hours after twelve successive unsuccessful call attempts.

Hong Kong

- For transmissions of any one document to any one telephone number, the application must make no more than eleven call attempts.

  There are no restrictions on the interval between each call attempt.
- The application must not attempt to transmit documents to 999 numbers.

Ireland

For calls to the same telephone number over the same trunk, the application must:

- Delay five seconds between the first and second call attempt.
- Delay sixty seconds between each subsequent call attempt.
- Make no more than four call attempts in one hour.

Israel

For transmissions of any one document to any one telephone number, the application must:

- Make no more than fifteen call attempts.
- Delay 30 seconds between each call attempt.

Italy

For calls to the same telephone number over the same trunk, the application must:

- Delay five seconds between the first and second call attempt.
- Delay sixty seconds between each subsequent call attempt.
- Make no more than four call attempts in one hour.

Japan

- For transmissions of any one document to any one telephone number, the application must do either of the following:
  - Make no more than three call attempts in a three minute period. New three minute periods begin three minutes after the beginning of the first attempt of the previous period.
  - Delay one minute between each call attempt, with no other restrictions.
- The application must not attempt to transmit documents to 119 or 110 numbers.
- If the application does not wait for dial tone, it must wait at least three seconds after seizing the line before dialing. Include `missing_wait 60` in *BT_CALL.CFG* to specify the three-second wait.

Malaysia

Pulse dialing is not supported in Malaysia.

For an unsuccessful attempt to dial any telephone number, the application must:

- Make no more than two additional call attempts.
- Delay a minimum of two minutes between each call attempt.

Netherlands

For transmissions of any one document to any one telephone number, the application must:

- Make no more than fifteen call attempts in one hour.
- Delay five seconds between the first and second call attempts.
- Delay one minute between each subsequent call attempt.

New Zealand

- For all outgoing calls, the application must:
  - Go on-hook for a minimum of five seconds between the end of one call and the beginning of the next call.
  - Clearly associate preprogrammed numbers with the names of the called parties and enable operators to easily modify the numbers.
- For transmissions of any one document to any one telephone number, the application must:
  - Make no more than five call attempts in one hour.
  - Make no more than a total of ten call attempts.
  - Delay sixty seconds between each call attempt.
- For calls to different telephone numbers over the same trunk, the application must do either of the following:
  - Delay sixty seconds between starting each call attempt.
  - Delay thirty seconds between each call attempt if it delays the next attempt three minutes after detecting the congestion tone (`ROBUSY`).

- For all incoming calls, the application must:
  - Delay from three to fifteen seconds from the detection of ringing before automatically answering a call.
  - Remain on-hook if the system has insufficient memory or disk space to perform its functions.

Norway

For calls to the same telephone number over the same trunk, the application must:

- Delay five seconds between the first and second call attempt.
- Delay sixty seconds between each subsequent call attempt.

Singapore

For an unsuccessful attempt to dial any telephone number, the application must:

- Make no more than ten additional call attempts.
- Delay a minimum of sixty seconds between each call attempt.

Spain

- For transmissions of any one document to any one telephone number, the application must:
  - Make no more than five call attempts in one hour.
  - Delay five seconds between the first and second call attempts.
  - Delay one minute between each subsequent call attempt.
- For calls to different telephone numbers over the same trunk, the application must delay two seconds between releasing the line after an unsuccessful call attempt and seizing it for the next call attempt.

Switzerland

- For an unsuccessful attempt to dial any telephone number, the application must:
  - Make no more than four call attempts. Each transmission of dialing information counts as one dial attempt.
  - Delay a minimum of thirty seconds between each call attempt.
  - Release the line for a minimum of five seconds before dialing a different telephone number.
- For a successful dialing to any telephone number, the application must:
  - Prevent automatic dial attempts to the same telephone number.
  - Release the line for a minimum of five seconds before dialing a different telephone number.

Turkey

For transmissions of any one document to any one telephone number, the application must:

- Make no more than fifteen call attempts.
- Delay one minute between each call attempt.

United Kingdom

For transmission of any one document to any one telephone number, the application must:

- Make no more than sixteen call attempts
- Delay five seconds between each call attempt.

United States

Unsuccessful calls are either unanswered calls or wrong calls. Unanswered calls occur when the called number fails to answer. In this case, the Bfv API reports `BUSY1`, `BUSY2`, `ROBUSY`, `RECALL`, `DIALTON`, `SIT_`, or `RNGNOANS`. Wrong calls occur when the called number answers but fails to send fax CED tone or V.21 signal to indicate a fax machine. In this case the Bfv API reports `HUMAN`, `QUIET`, `SILENCE`, `ANSWER`, or `G2DETCT`.

For transmission of any one document to any one telephone number:

- For unanswered calls, the application must:
  - Make no more than fifteen call attempts in one hour (unless the retry detector is manually reset).
  - Release the line within fifteen seconds after detection of one of the above call progress signals.
- For wrong calls, the application must:
  - Make no more than two call attempts in one hour (unless the retry detector is manually reset).
  - Release the line within fifteen seconds after detection of one of the above call progress signals.

Examples of R2 Parameter Files

The following R2 parameter file examples are country-specific. The country parameters files include:

- *Argentina R2 Parameter File on page 1459.*
- *Brazil R2 Parameter File on page 1462.*
- *China R2 Parameter File on page 1469.*
- *Korea R2 Parameter File on page 1472.*
- *Mexico R2 Parameter File on page 1475.*

Argentina R2 Parameter File

The following R2 parameter file example is used to configure IISDN R2-MFC for operation in Argentina.

```
/*********************************************************************
 *  r2Argentina.h - Argentina R2 parameters
 *  Description:
 *
 *  This include file contains the parameters needed to configure IISDN's
 *  R2-MFC for operation in Brazil.
 *
 *  Certain parameters are network specific and cannot be modified.
 *
 *  (C)-Copyright Dialogic Corporation. 2000
 ******************************************************************** /
#define R2_MF_PARAM_NOT_USED(0)/* clarify params not currently implemented */

static IISDN_E1_CAS_R2_DATA r2_Argentina =
{
// IISDN_R2_DIGITAL_LINE_SIG_PARAMS:
  200,  /* (unsigned short) r2OutSeizeTimer */
  90,   /* (unsigned short) r2OutAnswerTimeOut (s) */
  200,  /* (unsigned short) inboundReleaseGuardTime */
  0,    /* (unsigned short) inboundLineQualTimerIdle */
  0,    /* (unsigned short) DebugBitMask */

  1,    /* (unsigned char)  c_d_cas_bits ------> C=0 D=1 */
  1,    /* (unsigned char)  clearbackControl --> release guard / forced release */
```

```
/*
 * IISDN_R2_INTERREGISTER_PARAMS:
 *
 * NOTE!: Only the R2 in-dial application parameters may be modified by the
 user. All others are network specific and modification would likely cause
 malfunction.
 */
  7,      /* (unsigned short)dnisMaxNumDigits: max # of DNIS digits required */
  7,      /* (unsigned short)aniMaxNumDigits: max # of ANI digits required */
  1,      /* (unsigned short)dnisNumDigitsBeforeANI */

/*
 * R2 Inbound Timers
 */
  15,     /* (short) interForwardToneTimeOut: units in s or DISABLED */
  IISDN_INBOUND_PULSE_MIN_DELAY_ITU_MSEC, /* (short) prePulseToneDelay (s) */
  IISDN_INBOUND_PULSE_DURATION_ITU_MSEC,  /* (short) pulseToneDuration (s) */

/*
 * R2 Outbound Timers
 */
  17,     /* (short) forwardToneMaxOnTime: units in sec or DISABLED */
  27,     /* (short) forwardToneMaxOffTime: units in sec or DISABLED */
  17,     /* (short) forwardGroup2MaxOnTime: units in sec or DISABLED */

/*
 * R2 Backward Protocol Params
 */
  IISDN_INDIAL_DNIS_TIME_OUT_PULSE_GROUPII_REQUEST, /* (unsigned short)
    dnisTimeOutAction */

/*
 * Forward R2 Signals. Use IISDN_R2F_XX's
 */
  IISDN_R2F_INVALID, /* endOfDNIS */
  IISDN_R2F_15,      /* endOfANI_Available */
  IISDN_R2F_INVALID, /* halfEchoSuppressorRequired */
  IISDN_R2F_INVALID, /* noSatelliteLinkInCircuit */
  IISDN_R2F_INVALID, /* satelliteLinkInCircuit */
  IISDN_R2F_01,      /* aniCategoryDefault */

/*
 * Backward R2 signals. Use IISDN_R2B_XX's
 */
  IISDN_R2B_01,      /* sendNextDigitDNIS */
  IISDN_R2B_02,      /* sendLastButOneDigitDNIS */
  IISDN_R2B_03,      /* sendCallCategoryAndSwitchToGroupB_DNIS */
  IISDN_R2B_04,      /* congestion */
  IISDN_R2B_05,      /* sendCallingPartyCategory */
  IISDN_R2B_06,      /* callComplete_SetUpSpeechPath - Group A */
  IISDN_R2B_07,      /* sendLastButTwoDigitDNIS */
  IISDN_R2B_08,      /* sendLastButThreeDigitDNIS */
  IISDN_R2B_13,      /* requestNatureOfCircuit */
  IISDN_R2B_14,      /* requestIfHalfEchoSuppressorNeeded */
```

```
  IISDN_R2B_10,       /* sendFirstDigitDNIS */
  IISDN_R2B_05,       /* sendNextDigitANI */
  IISDN_R2B_INVALID, /* changeFrom_ANI_To_DNIS_SendNextDigit */
  IISDN_R2B_09,       /* changeFrom_ANI_To_DNIS_SendLastDigit */

    { /* GroupB_LineConditions */
      IISDN_R2B_06,       /* called Line Free, Charge */
      IISDN_R2B_07,       /* called Line Free, No Charge */
      IISDN_R2B_INVALID, /* alternate Line Free, Charge */
      IISDN_R2B_03,       /* called Line Busy */
      IISDN_R2B_08,       /* called Line Out Of Order */
      IISDN_R2B_05,       /* called Line Unallocated */
      IISDN_R2B_INVALID, /* spare */
      IISDN_R2B_INVALID, /* spare */
      IISDN_R2B_INVALID, /* spare */
      IISDN_R2B_INVALID, /* spare */
    },

/*
 * Call Progress generation
 */
  { /* IISDN_CPGEN_MF_PARAMS */
    { /* IISDN_CPGEN_MF_PARAMS for RING (cpSignals[0]) */
      425,  /* (unsigned short)freqTone1 */
      0,    /* (short) powerTone1 */
      0,    /* (unsigned short)freqTone2, NONE */
      0,    /* (short) powerTone2 */
      1,    /* (unsigned short)numCadences */
      1000, /* (short) makeTime1 */
      4000, /* (short) breakTime1 */
      0,    /* (short) makeTime2 */
      0,    /* (short) breakTime2 */
      0,    /* (short) makeTime3 */
      0,    /* (short) breakTime3 */
      2,    /* (unsigned short)numCycles, finite duration */
    },
    { /* IISDN_CPGEN_MF_PARAMS for BUSY (cpSignals[1]) */
      425,  /* (unsigned short)freqTone1 */
      0,    /* (short) powerTone1 */
      0,    /* (unsigned short)freqTone2, NONE */
      0,    /* (short) powerTone2 */
      1,    /* (unsigned short)numCadences */
      300,  /* (short) makeTime1 */
      200,  /* (short) breakTime1 */
      0,    /* (short) makeTime2 */
      0,    /* (short) breakTime2 */
      0,    /* (short) makeTime3 */
      0,    /* (short) breakTime3 */
      0,    /* (unsigned short)numCycles, repeat indefinitely */
    },
  },
};
```

Brazil R2 Parameter File

The following R2 parameter file example is used to configure IISDN R2-MFC for operation in Brazil.

```
/*********************************************************************
 *  r2Brazil.h - Brazil R2 parameters
 *  Description:
 *
 *  This include file contains the parameters needed to configure IISDN's
 *  R2-MFC for operation in Brazil.
 *
 *  Certain parameters are network specific and cannot be modified.
 *
 *  (C)-Copyright Dialogic Corporation. 2000
 ******************************************************************** /

#define R2_MF_PARAM_NOT_USED(0)/* clarify params not currently implemented */

static IISDN_E1_CAS_R2_DATA r2_Brazil =
{
// IISDN_R2_DIGITAL_LINE_SIG_PARAMS:
200,/* (unsigned short) r2OutSeizeTimer (ms) [100-200ms terrestrial,
 1-2 sec satellite */
60,/* (unsigned short) r2OutAnswerTimeOut (s) max time between Group B
 receipt and answer */

  0,  /* (unsigned short) reAnswerPulseDelay */
  0,  /* (unsigned short) reAnswerPulseLength */

  0,  /* (unsigned short) outboundClearbackQualTime */
  200,/* (unsigned short) inboundReleaseGuardTime */

  0,  /* (unsigned short) outboundDelayBeforeCallRequest */
  0,  /* (unsigned short) pad1 */

  0,  /* (unsigned char)  inboundLineQualTimerIdle */
  0,  /* (unsigned char)  inboundLineQualTimerCompelled */
  0,  /* (unsigned char)  inboundLineQualTimerConnected */
  0,  /* (unsigned char)  outboundLineQualTimerIdle */

  0,  /* (unsigned char)  outboundLineQualTimerCompelled */
  0,  /* (unsigned char)  outboundLineQualTimerConnected */
  0,  /* (unsigned char)  c_d_cas_bits */
  0,  /* (unsigned char)  metering_flags */

  0,  /* (unsigned char)  inboundClearbackControl */
  0,  /* (unsigned char)  detectBitFaultsDuringRing */
  0,  /* (unsigned char)  simultaneousBitTransitionWindow */
  1,  /* (unsigned char)  sendPreSeizeEvent */
```

```
/*
 * IISDN_R2_INTERREGISTER_PARAMS:
 *
 * NOTE!: Only the R2 in-dial application parameters may be modified by the
 user. All others are network specific and modification would likely cause
 malfunction.
 */
  7,      /* (unsigned short)dnisMinNumDigits: min # of DNIS digits required */
  7,      /* (unsigned short)dnisMaxNumDigits: max # of DNIS digits required */
  1,      /* (unsigned short)aniMinNumDigits: min # of ANI digits required */
  7,      /* (unsigned short)aniMaxNumDigits: max # of ANI digits required */
  IISDN_INDIAL_ADDR_COMPLETE_REQUEST_GROUP_B,/* (unsigned short)
    addressCompleteMode */
  IISDN_INDIAL_REPORT_ENSEMBLE_INFO,/* (unsigned short)infoReportMode */
  TRUE,   /* (unsigned short)enableVerificationDNIS: TRUE (1) or FALSE (0) */
  TRUE,   /* (unsigned short)enableVerificationANI: TRUE (1) or FALSE (0) */
  1,      /* (unsigned short)dnisNumDigitsBeforeANI */
  FALSE,  /* (unsigned short) aniRequestEnable */

  /*
   * DSP power references: Zero values entered below will enable
   * internal IISDN defaults....
   */
  1984,   /* (unsigned short) power0dBmInput:A-law setting for 0dBm RMS
    of a 1 kHz sine */
  1984,   /* (unsigned short) power0dBmOutput: A-law setting for 0dBm RMS
    of a 1 kHz sine */

  /*
   * R2 MF tone generation parameters.
   *
   *  (1) tone1Power: low frequency tone power, relative to system 0dBm level
   *      Units:0.5 dB
   *      Range:[3...-25]
   *      ITU (Q.454) Default:-16 (-8 dBm)
   *
   *  (2) tone2Power:high frequency tone power, relative to system 0dBm level
   *      Units:0.5 dB
   *      Range:[3...-25]
   *      ITU (Q.454) Default:-16 (-8 dBm)
   *
   */
  -16,/* (short) tone1Power */
  -16,/* (short) tone2Power */

  /*
   * R2 MF tone detection parameters. The zero values enable internal IISDN
   * defaults.
   *
   */
  0,  /* (short) minTonePower:*/
  0,  /* (unsigned short) minToneDuration:*/
  0,  /* (unsigned short) minSilence:*/
  0,  /* (unsigned short) freqTolerance:*/
```

```
/*
 * R2 Inbound Timers
 */
15,     /* (short) interForwardToneTimeOut: units in s or DISABLED */
IISDN_INBOUND_PULSE_MIN_DELAY_ITU_MSEC, /* (short) prePulseToneDelay (s) */
IISDN_INBOUND_PULSE_DURATION_ITU_MSEC,  /* (short) pulseToneDuration (s) */
IISDN_INBOUND_PULSE_NO_RECOGNITION_ITU_MSEC, /* (short)
  postPulseNoRecognitionDuration (s) */

/*
 * R2 Outbound Timers
 */
15,     /* (short) forwardToneMaxOnTime: units in sec or DISABLED */
27,     /* (short) forwardToneMaxOffTime: units in sec or DISABLED */
15,     /* (short) forwardGroup2MaxOnTime: units in sec or DISABLED */

/*
 * R2 Backward Protocol Params
 */
IISDN_INDIAL_DNIS_TIME_OUT_PULSE_GROUPII_REQUEST, /* (unsigned short)
  dnisTimeOutAction */
FALSE,/* (unsigned short)mustComplete_ANI_Flag: TRUE (1) or FALSE (0) */
R2_MF_PARAM_NOT_USED,/* (unsigned short) dnisNumberOfRepeats
  (NOT IMPLEMENTED) */

/*
 * Forward state machine. Fill in with an appropriate action from
 * IISDN_R2MFC_FORWARD_ACTIONS for each received signal code.
 */
{ /* processDNISForward[IISDN_NUM_R2MF_SIGNAL_CODES] (Group A) */
  IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,        /* action to R2B_INVALID */
  IISDN_PROCESS_NEXT_DNIS_DIGIT_REQUEST,        /* action to R2B_01 */
  IISDN_PROCESS_RESTART_DNIS_REQUEST,           /* action to R2B_02 */
  IISDN_PROCESS_CALL_COMPLETE_CHANGE_TO_GROUP_B,/* action to R2B_03 */
  IISDN_PROCESS_CONGESTION_SIGNAL,              /* action to R2B_04 */
  IISDN_PROCESS_CALLING_PARTY_CATEGORY_REQUEST, /* action to R2B_05 */
  IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,        /* action to R2B_06 */
  IISDN_PROCESS_LAST_BUT_2_DNIS_DIGIT_REQUEST,  /* action to R2B_07 */
  IISDN_PROCESS_LAST_BUT_3_DNIS_DIGIT_REQUEST,  /* action to R2B_08 */
  IISDN_PROCESS_LAST_BUT_1_DNIS_DIGIT_REQUEST,  /* action to R2B_09 */
  IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,        /* action to R2B_10 */
  IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,        /* action to R2B_11 */
  IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,        /* action to R2B_12 */
  IISDN_PROCESS_NATURE_OF_CIRCUIT_QUERY,        /* action to R2B_13 */
  IISDN_PROCESS_ECHO_SUPPRESSOR_QUERY,          /* action to R2B_14 */
  IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,        /* action to R2B_15 */
},
{ /* processANIForward[IISDN_NUM_R2MF_SIGNAL_CODES] (Group A or C) */
  IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,        /* action to R2B_INVALID */
  IISDN_PROCESS_NEXT_DNIS_DIGIT_REQUEST,        /* action to R2B_01 */
  IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,        /* action to R2B_02 */
  IISDN_PROCESS_CALL_COMPLETE_CHANGE_TO_GROUP_B,/* action to R2B_03 */
  IISDN_PROCESS_CONGESTION_SIGNAL,              /* action to R2B_04 */
  IISDN_PROCESS_NEXT_ANI_DIGIT_REQUEST,         /* action to R2B_05 */
  IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,        /* action to R2B_06 */
  IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,        /* action to R2B_07 */
```

```
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_08 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_09 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_10 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_11 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_12 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_13 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_14 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_15 */
  },
  { /* processCalledLineConditionForward[IISDN_NUM_R2MF_SIGNAL_CODES]
    (Group B) */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_INVALID */
    IISDN_PROCESS_GROUP_B_LINE_CONDITION,     /* action to R2B_01 */
    IISDN_PROCESS_GROUP_B_LINE_CONDITION,     /* action to R2B_02 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_03 */
    IISDN_PROCESS_GROUP_B_LINE_CONDITION,     /* action to R2B_04 */
    IISDN_PROCESS_GROUP_B_LINE_CONDITION,     /* action to R2B_05 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_06 */
    IISDN_PROCESS_GROUP_B_LINE_CONDITION,     /* action to R2B_07 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_08 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_09 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_10 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_11 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_12 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_13 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_14 */
    IISDN_PROCESS_INVALID_BACKWARD_SIGNAL,    /* action to R2B_15 */
  },

  /*
   * Backward state machine. Fill in with an appropriate action from
   * IISDN_R2MFC_BACKWARD_ACTIONS for each received signal code.
   */
  { /* processDNISBackward[IISDN_NUM_R2MF_SIGNAL_CODES] (Group I) */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,     /* action to R2F_INVALID */
    IISDN_PROCESS_DNIS_DIGIT,                 /* action to R2F_01 */
    IISDN_PROCESS_DNIS_DIGIT,                 /* action to R2F_02 */
    IISDN_PROCESS_DNIS_DIGIT,                 /* action to R2F_03 */
    IISDN_PROCESS_DNIS_DIGIT,                 /* action to R2F_04 */
    IISDN_PROCESS_DNIS_DIGIT,                 /* action to R2F_05 */
    IISDN_PROCESS_DNIS_DIGIT,                 /* action to R2F_06 */
    IISDN_PROCESS_DNIS_DIGIT,                 /* action to R2F_07 */
    IISDN_PROCESS_DNIS_DIGIT,                 /* action to R2F_08 */
    IISDN_PROCESS_DNIS_DIGIT,                 /* action to R2F_09 */
    IISDN_PROCESS_DNIS_DIGIT,                 /* action to R2F_10 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,     /* action to R2F_11 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,     /* action to R2F_12 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,     /* action to R2F_13 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,     /* action to R2F_14 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,     /* action to R2F_15 */
  },
  { /* processCallCategoryBackward[IISDN_NUM_R2MF_SIGNAL_CODES] (Group IIa) */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_INVALID */
    IISDN_PROCESS_CALL_CATEGORY_AND_SEND_LINE_STATE,/* action to R2F_01 */
    IISDN_PROCESS_CALL_CATEGORY_AND_SEND_LINE_STATE,/* action to R2F_02 */
    IISDN_PROCESS_CALL_CATEGORY_AND_SEND_LINE_STATE,/* action to R2F_03 */
```

```
    IISDN_PROCESS_CALL_CATEGORY_AND_SEND_LINE_STATE,/* action to R2F_04 */
    IISDN_PROCESS_CALL_CATEGORY_AND_SEND_LINE_STATE,/* action to R2F_05 */
    IISDN_PROCESS_CALL_CATEGORY_AND_SEND_LINE_STATE,/* action to R2F_06 */
    IISDN_PROCESS_CALL_CATEGORY_AND_SEND_LINE_STATE,/* action to R2F_07 */
    IISDN_PROCESS_CALL_CATEGORY_AND_SEND_LINE_STATE,/* action to R2F_08 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_09 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_10 */
    IISDN_PROCESS_CALL_CATEGORY_AND_SEND_LINE_STATE,/* action to R2F_11 */
    IISDN_PROCESS_CALL_CATEGORY_AND_SEND_LINE_STATE,/* action to R2F_12 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_13 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_14 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_15 */
  },
  { /* processANIBackward[IISDN_NUM_R2MF_SIGNAL_CODES] (Group I or III) */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL, /* action to R2F_INVALID */
    IISDN_PROCESS_ANI_DIGIT,              /* action to R2F_01 */
    IISDN_PROCESS_ANI_DIGIT,              /* action to R2F_02 */
    IISDN_PROCESS_ANI_DIGIT,              /* action to R2F_03 */
    IISDN_PROCESS_ANI_DIGIT,              /* action to R2F_04 */
    IISDN_PROCESS_ANI_DIGIT,              /* action to R2F_05 */
    IISDN_PROCESS_ANI_DIGIT,              /* action to R2F_06 */
    IISDN_PROCESS_ANI_DIGIT,              /* action to R2F_07 */
    IISDN_PROCESS_ANI_DIGIT,              /* action to R2F_08 */
    IISDN_PROCESS_ANI_DIGIT,              /* action to R2F_09 */
    IISDN_PROCESS_ANI_DIGIT,              /* action to R2F_10 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL, /* action to R2F_11 */
    IISDN_PROCESS_ANI_NOT_AVAILABLE,      /* action to R2F_12 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL, /* action to R2F_13 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL, /* action to R2F_14 */
    IISDN_PROCESS_ANI_END_OF_ID,          /* action to R2F_15 */
  },
  { /* processCallingPartyCategoryBackward[IISDN_NUM_R2MF_SIGNAL_CODES]
 (Group IIb) */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_INVALID */
    IISDN_PROCESS_CALLING_CATEGORY_AND_REQUEST_ANI, /* action to R2F_01 */
    IISDN_PROCESS_CALLING_CATEGORY_AND_REQUEST_ANI, /* action to R2F_02 */
    IISDN_PROCESS_CALLING_CATEGORY_AND_REQUEST_ANI, /* action to R2F_03 */
    IISDN_PROCESS_CALLING_CATEGORY_AND_REQUEST_ANI, /* action to R2F_04 */
    IISDN_PROCESS_CALLING_CATEGORY_AND_REQUEST_ANI, /* action to R2F_05 */
    IISDN_PROCESS_CALLING_CATEGORY_AND_REQUEST_ANI, /* action to R2F_06 */
    IISDN_PROCESS_CALLING_CATEGORY_AND_REQUEST_ANI, /* action to R2F_07 */
    IISDN_PROCESS_CALLING_CATEGORY_AND_REQUEST_ANI, /* action to R2F_08 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_09 */
    IISDN_PROCESS_CALLING_CATEGORY_AND_REQUEST_ANI, /* action to R2F_10 */
    IISDN_PROCESS_CALLING_CATEGORY_AND_REQUEST_ANI, /* action to R2F_11 */
    IISDN_PROCESS_CALLING_CATEGORY_REQUEST_DENIED,  /* action to R2F_12 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_13 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_14 */
    IISDN_PROCESS_INVALID_FORWARD_SIGNAL,           /* action to R2F_15 */
  },
```

```
/*
   * Forward R2 Signals. Use IISDN_R2F_XX's
   */
  IISDN_R2F_INVALID,   /* endOfDNIS */
  IISDN_R2F_15,        /* endOfANI_Available */
  IISDN_R2F_INVALID,   /* endOfANI_Restricted */
  IISDN_R2F_12,        /* callingPartyCategoryRequestNotAccepted */
  IISDN_R2F_12,        /* aniRequestNotAccepted */
  IISDN_R2F_14,        /* halfEchoSuppressorRequired */
  IISDN_R2F_13,        /* noSatelliteLinkInCircuit */
  IISDN_R2F_14,        /* satelliteLinkInCircuit */
  IISDN_R2F_01,        /* aniCategoryDefault */
  IISDN_R2F_01,        /* dnisCategoryDefault */

  /*
   * Backward R2 signals. Use IISDN_R2B_XX's
   */
  IISDN_R2B_01,        /* sendNextDigitDNIS */
  IISDN_R2B_09,        /* sendLastButOneDigitDNIS */
  IISDN_R2B_03,        /* sendCallCategoryAndSwitchToGroupB_DNIS */
  IISDN_R2B_04,        /* congestion_DNIS */
  IISDN_R2B_05,        /* sendCallingPartyCategory */
  IISDN_R2B_INVALID,   /* callComplete_SetUpSpeechPath_DNIS */
  IISDN_R2B_07,        /* sendLastButTwoDigitDNIS */
  IISDN_R2B_08,        /* sendLastButThreeDigitDNIS */
  IISDN_R2B_13,        /* requestNatureOfCircuit */
  IISDN_R2B_14,        /* requestIfHalfEchoSuppressorNeeded */
  IISDN_R2B_02,        /* sendFirstDigitDNIS */
  IISDN_R2B_05,        /* sendNextDigitANI */
  IISDN_R2B_03,        /* sendCallCategoryAndSwitchToGroupB_ANI */
  IISDN_R2B_04,        /* congestion_ANI */
  IISDN_R2B_01,        /* changeFrom_ANI_To_DNIS_SendNextDigit */
  IISDN_R2B_INVALID,   /* changeFrom_ANI_To_DNIS_SendLastDigit */
  IISDN_R2B_INVALID,   /* changeFrom_ANI_To_DNIS_SendFirstDigit */
  IISDN_R2B_INVALID,   /* callComplete_SetUpSpeechPath_ANI */
  IISDN_R2B_01,        /* groupB_CalledLineConditionDefault */
  IISDN_R2B_INVALID,   /* pad1 */

    { /* GroupB_LineConditions */
      IISDN_R2B_01,      /* called Line Free, Charge */
      IISDN_R2B_05,      /* called Line Free, No Charge */
      IISDN_R2B_INVALID, /* user Defined Called Line Condition3 */
      IISDN_R2B_02,      /* called Line Busy */
      IISDN_R2B_INVALID, /* called Line Out Of Order */
      IISDN_R2B_07,      /* called Line Unallocated */
      IISDN_R2B_04,      /* called Line Congested */
      IISDN_R2B_INVALID, /* called Line Last Party Released */
      IISDN_R2B_INVALID, /* user Defined Called Line Condition1 */
      IISDN_R2B_INVALID, /* user Defined Called Line Condition2 */
      IISDN_R2B_INVALID, /* user Defined Called Line Condition3 */
      IISDN_R2B_INVALID, /* user Defined Called Line Condition4 */
      IISDN_R2B_INVALID, /* user Defined Called Line Condition5 */
      IISDN_R2B_INVALID, /* user Defined Called Line Condition6 */
      IISDN_R2B_INVALID, /* user Defined Called Line Condition7 */
      IISDN_R2B_INVALID, /* user Defined Called Line Condition8 */
    },
```

```
  0,  /* halfEchoSuppressorFlag */
  0,  /* pad2 */

  0xC,/* (unsigned short)r2mfcTraceEnable */

    /*
   * Call Progress generation
   */
  2,/* (unsigned short)numCallProgSignalsDefined; max number defined for
 R2 is 2! */
  { /* IISDN_CPGEN_MF_PARAMS */
    { /* IISDN_CPGEN_MF_PARAMS for RING (cpSignals[0]) */
      425,    /* (unsigned short)freqTone1 */
      -20,    /* (short) powerTone1, -10 dBm */
      0,      /* (unsigned short)freqTone2, NONE */
      -20,    /* (short) powerTone2 */
      1,      /* (unsigned short)numCadences */
      1000,   /* (short) makeTime1 */
      4000,   /* (short) breakTime1 */
      0,      /* (short) makeTime2 */
      0,      /* (short) breakTime2 */
      0,      /* (short) makeTime3 */
      0,      /* (short) breakTime3 */
      2,      /* (unsigned short)numCycles, finite duration */
    },
    { /* IISDN_CPGEN_MF_PARAMS for BUSY (cpSignals[1]) */
      425,    /* (unsigned short)freqTone1 */
      -20,    /* (short) powerTone1, -10 dBm */
      0,      /* (unsigned short)freqTone2, NONE */
      -20,    /* (short) powerTone2 */
      1,      /* (unsigned short)numCadences */
      250,    /* (short) makeTime1 */
      250,    /* (short) breakTime1 */
      0,      /* (short) makeTime2 */
      0,      /* (short) breakTime2 */
      0,      /* (short) makeTime3 */
      0,      /* (short) breakTime3 */
      0,      /* (unsigned short)numCycles, repeat indefinitely */
    },
  },
};
```

China R2 Parameter File

The following R2 parameter file example is used to configure IISDN R2-MFC for operation in China.

```
/*********************************************************************
 *  r2China.h - China R2 parameters
 *
 *  Description:
 *
 *  This include file contains the parameters needed to configure IISDN's
 *  R2-MFC for operation in China.
 *
 *  Certain parameters are network specific and cannot be modified.
 *
 *  (C) - Copyright Dialogic Corporation. 2000
 *********************************************************************/

#define R2_MF_PARAM_NOT_USED(0)/* clarify params not currently implemented */

static IISDN_E1_CAS_R2_DATA r2_China =
{
// IISDN_R2_DIGITAL_LINE_SIG_PARAMS:
  200,/* (unsigned short) r2OutSeizeTimer */
  90, /* (unsigned short) r2OutAnswerTimeOut (s) */
  200,/* (unsigned short) inboundReleaseGuardTime */
  0,  /* (unsigned short) inboundLineQualTimerIdle */
  0,  /* (unsigned short) DebugBitMask */

  3,  /* (unsigned char)  c_d_cas_bits */
  0,  /* (unsigned char)  clearbackControl ----> release guard / forced release */

/*
 * IISDN_R2_INTERREGISTER_PARAMS:
 *
 * NOTE!: Only the R2 in-dial application parameters may be modified by the
 user. All others are network specific and modification would likely cause
 malfunction.
 */
  7,  /* (unsigned short)dnisMaxNumDigits: max # of DNIS digits required */
  7,  /* (unsigned short)aniMaxNumDigits: max # of ANI digits required */
  1,  /* (unsigned short)dnisNumDigitsBeforeANI */

  /*
   * R2 Inbound Timers
   */
  15,     /* (short) interForwardToneTimeOut: units in s or DISABLED */
  IISDN_INBOUND_PULSE_MIN_DELAY_ITU_MSEC, /* (short) prePulseToneDelay (s) */
  IISDN_INBOUND_PULSE_DURATION_ITU_MSEC,  /* (short) pulseToneDuration (s) */
```

```
  /*
   * R2 Outbound Timers
   */
  17, /* (short) forwardToneMaxOnTime: units in sec or DISABLED */
  27, /* (short) forwardToneMaxOffTime: units in sec or DISABLED */
  17, /* (short) forwardGroup2MaxOnTime: units in sec or DISABLED */

  /*
   * R2 Backward Protocol Params
   */
  IISDN_INDIAL_DNIS_TIME_OUT_PULSE_GROUPII_REQUEST, /* (unsigned short)
    dnisTimeOutAction */

/*
   * Forward R2 Signals. Use IISDN_R2F_XX's
   */
  IISDN_R2F_INVALID,   /* endOfDNIS */
  IISDN_R2F_15,        /* endOfANI_Available */
  IISDN_R2F_INVALID,   /* halfEchoSuppressorRequired */
  IISDN_R2F_INVALID,   /* noSatelliteLinkInCircuit */
  IISDN_R2F_INVALID,   /* satelliteLinkInCircuit */
  IISDN_R2F_01,        /* aniCategoryDefault */

  /*
   * Backward R2 signals. Use IISDN_R2B_XX's
   */
  IISDN_R2B_01,        /* sendNextDigitDNIS */
  IISDN_R2B_INVALID,   /* sendLastButOneDigitDNIS */
  IISDN_R2B_03,        /* sendCallCategoryAndSwitchToGroupB_DNIS */
  IISDN_R2B_04,        /* congestion */
  IISDN_R2B_06,        /* sendCallingPartyCategory */
  IISDN_R2B_INVALID,   /* callComplete_SetUpSpeechPath - Group A */
  IISDN_R2B_INVALID,   /* sendLastButTwoDigitDNIS */
  IISDN_R2B_INVALID,   /* sendLastButThreeDigitDNIS */
  IISDN_R2B_INVALID,   /* requestNatureOfCircuit */
  IISDN_R2B_INVALID,   /* requestIfHalfEchoSuppressorNeeded */
  IISDN_R2B_02,        /* sendFirstDigitDNIS */
  IISDN_R2B_01,        /* sendNextDigitANI */
  IISDN_R2B_INVALID,   /* changeFrom_ANI_To_DNIS_SendNextDigit */
  IISDN_R2B_INVALID,   /* changeFrom_ANI_To_DNIS_SendLastDigit */

    { /* GroupB_LineConditions */
      IISDN_R2B_01,        /* called Line Free, Charge */
      IISDN_R2B_INVALID,   /* called Line Free, No Charge */
      IISDN_R2B_INVALID,   /* alternate called Line Free, Charge */
      IISDN_R2B_02,        /* called Line Busy */
      IISDN_R2B_INVALID,   /* called Line Out Of Order */
      IISDN_R2B_05,        /* called Line Unallocated */
      IISDN_R2B_INVALID,   /* spare */
      IISDN_R2B_INVALID,   /* spare */
      IISDN_R2B_INVALID,   /* spare */
      IISDN_R2B_INVALID,   /* spare */
    },
```

```
  /*
   * Call Progress generation
   */
  { /* IISDN_CPGEN_MF_PARAMS */
    { /* IISDN_CPGEN_MF_PARAMS for RING (cpSignals[0]) */
      450,  /* (unsigned short)freqTone1 */
      0,    /* (short) powerTone1, -10 dBm */
      0,    /* (unsigned short)freqTone2, NONE */
      0,    /* (short) powerTone2 */
      1,    /* (unsigned short)numCadences */
      1000, /* (short) makeTime1 */
      4000, /* (short) breakTime1 */
      0,    /* (short) makeTime2 */
      0,    /* (short) breakTime2 */
      0,    /* (short) makeTime3 */
      0,    /* (short) breakTime3 */
      2,    /* (unsigned short)numCycles, finite duration */
    },
    { /* IISDN_CPGEN_MF_PARAMS for BUSY (cpSignals[1]) */
      450,  /* (unsigned short)freqTone1 */
      0,    /* (short) powerTone1, -10 dBm */
      0,    /* (unsigned short)freqTone2, NONE */
      0,    /* (short) powerTone2 */
      1,    /* (unsigned short)numCadences */
      350,  /* (short) makeTime1 */
      350,  /* (short) breakTime1 */
      0,    /* (short) makeTime2 */
      0,    /* (short) breakTime2 */
      0,    /* (short) makeTime3 */
      0,    /* (short) breakTime3 */
      0,    /* (unsigned short)numCycles, repeat indefinitely */
    },
  },
};
```

Korea R2 Parameter File

The following R2 parameter file example is used to configure IISDN R2-MFC for operation in Korea.

```
/*********************************************************************
 *  r2Korea.h - Korea R2 parameters
 *
 *  Description:
 *
 *  This include file contains the parameters needed to configure IISDN's
 *  R2-MFC for operation in Korea.
 *
 *  Certain parameters are network specific and cannot be modified.
 *
 *  (C) - Copyright Dialogic Corporation 2000
 *********************************************************************/

#define R2_MF_PARAM_NOT_USED(0)/* clarify params not currently implemented */

static IISDN_E1_CAS_R2_DATA r2_Korea =
{
// IISDN_R2_DIGITAL_LINE_SIG_PARAMS:
  200,/* (unsigned short) r2OutSeizeTimer (ms) */
  90, /* (unsigned short) r2OutAnswerTimeOut */
  0,  /* (unsigned short) inboundReleaseGuardTime */
  100,/* (unsigned short) inboundLineQualTimerIdle */
  0,  /* (unsigned short) DebugBitMask */

  1,  /* (unsigned char)  c_d_cas_bits ------> C=0 D=1 */
  0,  /* (unsigned char)  clearbackControl --> release guard / forced release */

/*
 * IISDN_R2_INTERREGISTER_PARAMS:
 *
 * NOTE!: Only the R2 in-dial application parameters may be modified by the
 user. All others are network specific and modification would likely cause
 malfunction.
 */
  7,  /* (unsigned short)dnisMaxNumDigits: max # of DNIS digits required */
  7,  /* (unsigned short)aniMaxNumDigits: max # of ANI digits required */
  1,  /* (unsigned short)dnisNumDigitsBeforeANI */

  /*
   * R2 Inbound Timers
   */
  15, /* (short) interForwardToneTimeOut: units in s or DISABLED */
  IISDN_INBOUND_PULSE_MIN_DELAY_ITU_MSEC, /* (short) prePulseToneDelay (s) */
  IISDN_INBOUND_PULSE_DURATION_ITU_MSEC,  /* (short) pulseToneDuration (s) */
```

```
  /*
   * R2 Outbound Timers
   */
  17, /* (short) forwardToneMaxOnTime: units in sec or DISABLED */
  27, /* (short) forwardToneMaxOffTime: units in sec or DISABLED */
  17, /* (short) forwardGroup2MaxOnTime: units in sec or DISABLED */

  /*
   * R2 Backward Protocol Params
   */
  IISDN_INDIAL_DNIS_TIME_OUT_PULSE_GROUPII_REQUEST, /* (unsigned short)
    dnisTimeOutAction */

/*
   * Forward R2 Signals. Use IISDN_R2F_XX's
   */
  IISDN_R2F_15,       /* endOfDNIS */
  IISDN_R2F_15,       /* endOfANI_Available */
  IISDN_R2F_INVALID, /* halfEchoSuppressorRequired */
  IISDN_R2F_INVALID, /* noSatelliteLinkInCircuit */
  IISDN_R2F_INVALID, /* satelliteLinkInCircuit */
  IISDN_R2F_01,       /* aniCategoryDefault */

  /*
   * Backward R2 signals. Use IISDN_R2B_XX's
   */
  IISDN_R2B_01,       /* sendNextDigitDNIS */
  IISDN_R2B_02,       /* sendLastButOneDigitDNIS */
  IISDN_R2B_03,       /* sendCallCategoryAndSwitchToGroupB_DNIS */
  IISDN_R2B_04,       /* congestion */
  IISDN_R2B_05,       /* sendCallingPartyCategory */
  IISDN_R2B_06,       /* callComplete_SetUpSpeechPath - Group A*/
  IISDN_R2B_07,       /* sendLastButTwoDigitDNIS */
  IISDN_R2B_08,       /* sendLastButThreeDigitDNIS */
  IISDN_R2B_13,       /* requestNatureOfCircuit */
  IISDN_R2B_14,       /* requestIfHalfEchoSuppressorNeeded */
  IISDN_R2B_09,       /* sendFirstDigitDNIS */
  IISDN_R2B_05,       /* sendNextDigitANI */
  IISDN_R2B_INVALID, /* changeFrom_ANI_To_DNIS_SendNextDigit */
  IISDN_R2B_INVALID, /* changeFrom_ANI_To_DNIS_SendLastDigit */

    { /* GroupB_LineConditions */
      IISDN_R2B_06,       /* called Line Free, Charge */
      IISDN_R2B_07,       /* called Line Free, No Charge */
      IISDN_R2B_01,       /* alternate called Line Free, Charge */
      IISDN_R2B_03,       /* called Line Busy */
      IISDN_R2B_08,       /* called Line Out Of Order */
      IISDN_R2B_05,       /* called Line Unallocated */
      IISDN_R2B_INVALID, /* spare */
      IISDN_R2B_INVALID, /* spare */
      IISDN_R2B_INVALID, /* spare */
      IISDN_R2B_INVALID, /* spare */
    },
```

```
  /*
   * Call Progress generation
   */
  { /* IISDN_CPGEN_MF_PARAMS */
    { /* IISDN_CPGEN_MF_PARAMS for RING (cpSignals[0]) */
      440,  /* (unsigned short)freqTone1 */
      0,    /* (short) powerTone1 */
      480,  /* (unsigned short)freqTone2, NONE */
      0,    /* (short) powerTone2 */
      1,    /* (unsigned short)numCadences */
      1000, /* (short) makeTime1 */
      4000, /* (short) breakTime1 */
      0,    /* (short) makeTime2 */
      0,    /* (short) breakTime2 */
      0,    /* (short) makeTime3 */
      0,    /* (short) breakTime3 */
      2,    /* (unsigned short)numCycles, finite duration */
    },
    { /* IISDN_CPGEN_MF_PARAMS for BUSY (cpSignals[1]) */
      440,  /* (unsigned short)freqTone1 */
      0,    /* (short) powerTone1 */
      480,  /* (unsigned short)freqTone2, NONE */
      0,    /* (short) powerTone2 */
      1,    /* (unsigned short)numCadences */
      500,  /* (short) makeTime1 */
      500,  /* (short) breakTime1 */
      0,    /* (short) makeTime2 */
      0,    /* (short) breakTime2 */
      0,    /* (short) makeTime3 */
      0,    /* (short) breakTime3 */
      0,    /* (unsigned short)numCycles, repeat indefinitely */
    },
  },
};
```

Mexico R2 Parameter File

The following R2 parameter file example is used to configure IISDN R2-MFC for operation in Mexico.

```
/*********************************************************************
 *  r2Mexico.h - Mexico R2 parameters
 *
 *  Description:
 *
 *  This include file contains the parameters needed to configure IISDN's
 *  R2-MFC for operation in Mexico.
 *
 *  Certain parameters are network specific and cannot be modified.
 *
 *  (C)-Copyright Dialogic Corporation 2000
 *********************************************************************/

#define R2_MF_PARAM_NOT_USED(0)/* clarify params not currently implemented */

static IISDN_E1_CAS_R2_DATA r2_Mexico =
{
// IISDN_R2_DIGITAL_LINE_SIG_PARAMS:
  200,/* (unsigned short) r2OutSeizeTimer (ms) */
  90, /* (unsigned short) r2OutAnswerTimeOut (s) */
  200,/* (unsigned short) inboundReleaseGuardTime */
  0,  /* (unsigned short) inboundLineQualTimerIdle */
  0,  /* (unsigned short) DebugBitMask */

  1,  /* (unsigned char)  c_d_cas_bits ------> C=0 D=1 */
  0,  /* (unsigned char)  clearbackControl ----> release guard / forced release */

/*
 * IISDN_R2_INTERREGISTER_PARAMS:
 *
 * NOTE!: Only the R2 in-dial application parameters may be modified by the
 user. All others are network specific and modification would likely cause
 malfunction.
 */
  7,  /* (unsigned short)dnisMaxNumDigits: max # of DNIS digits required */
  7,  /* (unsigned short)aniMaxNumDigits: max # of ANI digits required */
  1,  /* (unsigned short)dnisNumDigitsBeforeANI */

  /*
   * R2 Inbound Timers
   */
  11,     /* (short) interForwardToneTimeOut: units in s or DISABLED */
  IISDN_INBOUND_PULSE_MIN_DELAY_ITU_MSEC, /* (short) prePulseToneDelay (s) */
  IISDN_INBOUND_PULSE_DURATION_ITU_MSEC,  /* (short) pulseToneDuration (s) */
```

```
  /*
   * R2 Outbound Timers
   */
  17,   /* (short) forwardToneMaxOnTime: units in sec or DISABLED */
  27,   /* (short) forwardToneMaxOffTime: units in sec or DISABLED */
  17,   /* (short) forwardGroup2MaxOnTime: units in sec or DISABLED */

  /*
   * R2 Backward Protocol Params
   */
  IISDN_INDIAL_DNIS_TIME_OUT_PULSE_GROUPII_REQUEST, /* (unsigned short)
    dnisTimeOutAction */

/*
   * Forward R2 Signals. Use IISDN_R2F_XX's
   */
  IISDN_R2F_INVALID, /* endOfDNIS */
  IISDN_R2F_15,      /* endOfANI_Available */
  IISDN_R2F_INVALID, /* halfEchoSuppressorRequired */
  IISDN_R2F_INVALID, /* noSatelliteLinkInCircuit */
  IISDN_R2F_INVALID, /* satelliteLinkInCircuit */
  IISDN_R2F_02,      /* aniCategoryDefault */

  /*
   * Backward R2 signals. Use IISDN_R2B_XX's
   */
  IISDN_R2B_01,      /* sendNextDigitDNIS */
  IISDN_R2B_INVALID, /* sendLastButOneDigitDNIS */
  IISDN_R2B_03,      /* sendCallCategoryAndSwitchToGroupB_DNIS */
  IISDN_R2B_04,      /* congestion */
  IISDN_R2B_06,      /* sendCallingPartyCategory */
  IISDN_R2B_INVALID, /* callComplete_SetUpSpeechPath - Group A*/
  IISDN_R2B_INVALID, /* sendLastButTwoDigitDNIS */
  IISDN_R2B_INVALID, /* sendLastButThreeDigitDNIS */
  IISDN_R2B_INVALID, /* requestNatureOfCircuit */
  IISDN_R2B_INVALID, /* requestIfHalfEchoSuppressorNeeded */
  IISDN_R2B_02,      /* sendFirstDigitDNIS */
  IISDN_R2B_01,      /* sendNextDigitANI */
  IISDN_R2B_05,      /* changeFrom_ANI_To_DNIS_SendNextDigit */
  IISDN_R2B_06,      /* changeFrom_ANI_To_DNIS_SendLastDigit */

    { /* GroupB_LineConditions */
      IISDN_R2B_01,      /* called Line Free, Charge */
      IISDN_R2B_05,      /* called Line Free, No Charge */
      IISDN_R2B_INVALID, /* alternate called Line Free, Charge */
      IISDN_R2B_02,      /* called Line Busy */
      IISDN_R2B_INVALID, /* called Line Out Of Order */
      IISDN_R2B_INVALID, /* called Line Unallocated */
      IISDN_R2B_INVALID, /* spare */
      IISDN_R2B_INVALID, /* spare */
      IISDN_R2B_INVALID, /* spare */
      IISDN_R2B_INVALID, /* spare */
    },
```

```
  /*
   * Call Progress generation
   */
  { /* IISDN_CPGEN_MF_PARAMS */
    { /* IISDN_CPGEN_MF_PARAMS for RING (cpSignals[0]) */
      425,  /* (unsigned short)freqTone1 */
      0,    /* (short) powerTone1 */
      0,    /* (unsigned short)freqTone2, NONE */
      0,    /* (short) powerTone2 */
      1,    /* (unsigned short)numCadences */
      1000, /* (short) makeTime1 */
      4000, /* (short) breakTime1 */
      0,    /* (short) makeTime2 */
      0,    /* (short) breakTime2 */
      0,    /* (short) makeTime3 */
      0,    /* (short) breakTime3 */
      2,    /* (unsigned short)numCycles, finite duration */
    },
    { /* IISDN_CPGEN_MF_PARAMS for BUSY (cpSignals[1]) */
      425,  /* (unsigned short)freqTone1 */
      0,    /* (short) powerTone1 */
      0,    /* (unsigned short)freqTone2, NONE */
      0,    /* (short) powerTone2 */
      1,    /* (unsigned short)numCadences */
      250,  /* (short) makeTime1 */
      250,  /* (short) breakTime1 */
      0,    /* (short) makeTime2 */
      0,    /* (short) breakTime2 */
      0,    /* (short) makeTime3 */
      0,    /* (short) breakTime3 */
      0,    /* (unsigned short)numCycles, repeat indefinitely */
    },
  },
};
```

H - Unsupported Functionality

The following is unsupported functionality in the SDK:

- TR1000 advanced speech related functionality
  - Dialogic® Brooktrout® TR1000 media boards ("TR1000 boards")
  - Conferencing functionality (BfvCallConferencexxxx)
  - Full duplex speech functionality.
  - Accucall
- QSIG Call Control protocol
  - BfvCallDivert()
  - BfvCallWaitForDivert()
  - BfvLineDivert()
- OSI Library
- Input Fields for BfvCallSWConnectIP function

The following IPV4-only input fields for the BrvCallSWConnectIP function are no longer supported and have been replaced with input fields that support both IPV4 and IPV6.

```
short             destOptions.RTPopts.localRTPAddr.sin_family;
unsigned short    destOptions.RTPopts.localRTPAddr.sin_port;
unsigned          destOptions.RTPopts.localRTPAddr.sin_addr.S_un.S_addr;
short             destOptions.RTPopts.localRTCPAddr.sin_family;
unsigned short    destOptions.RTPopts.localRTCPAddr.sin_port;
unsigned          destOptions.RTPopts.localRTCPAddr.sin_addr.S_un.S_addr;
short             destOptions.RTPopts.remoteRTPAddr.sin_family;
unsigned short    destOptions.RTPopts.remoteRTPAddr.sin_port;
unsigned          destOptions.RTPopts.remoteRTPAddr.sin_addr.S_un.S_addr;
short             destOptions.RTPopts.remoteRTCPAddr.sin_family;
unsigned short    destOptions.RTPopts.remoteRTCPAddr.sin_port;
unsigned          destOptions.RTPopts.remoteRTCPAddr.sin_addr.S_un.S_addr;
short             destOptions.UDPTLopts.localAddr.sin_family;
unsigned short    destOptions.UDPTLopts.localAddr.sin_port;
unsigned          destOptions.UDPTLopts.localAddr.sin_addr.S_un.S_addr;
short             destOptions.UDPTLopts.remoteAddr.sin_family;
unsigned short    destOptions.UDPTLopts.remoteAddr.sin_port;
unsigned          destOptions.UDPTLopts.remoteAddr.sin_addr.S_un.S_addr;
```

args.destOptions.RTPopts.localRTPAddr.sin_family

This indicates the type of local RTP addressing.

Valid values are:

TELE_CTRL_AF_INET_DEF

args.destOptions.RTPopts.localRTPAddr.sin_port

This indicates the local RTP port number.

args.destOptions.RTPopts.localRTPAddr.sin_addr.S_un.S_addr

This indicates the local RTP IP address as an integer where

Class A is byte 4

Class B is byte 3

Class C is byte 2

Class D is byte 1

Example: 10.128.100.100 would be 0x0A806464

args.destOptions.RTPopts.localRTCPAddr.sin_family

This indicates the type of local RTCP addressing.

Valid values are:

TELE_CTRL_AF_INET_DEF

args.destOptions.RTPopts.localRTCPAddr.sin_port

This indicates the local RTCP port number.

args.destOptions.RTPopts.localRTCPAddr.sin_addr.S_un.S_addr

This indicates the local RTCP IP address as an integer where

Class A is byte 4

Class B is byte 3

Class C is byte 2

Class D is byte 1

Example: 10.128.100.100 would be 0x0A806464

args.destOptions.RTPopts.remoteRTPAddr.sin_family

This indicates the type of remote RTP addressing.

Valid values are:

TELE_CTRL_AF_INET_DEF

args.destOptions.RTPopts.remoteRTPAddr.sin_port

This indicates the remote RTP port number.

args.destOptions.RTPopts.remoteRTPAddr.sin_addr.S_un.S_addr

This indicates the remote RTP IP address as an integer where

Class A is byte 4

Class B is byte 3

Class C is byte 2

Class D is byte 1

Example: 10.128.100.100 would be 0x0A806464

args.destOptions.RTPopts.remoteRTCPAddr.sin_family

This indicates the type of remote RTCP addressing.

Valid values are:

TELE_CTRL_AF_INET_DEF

args.destOptions.RTPopts.remoteRTCPAddr.sin_port

This indicates the remote RTCP port number.

args.destOptions.RTPopts.remoteRTCPAddr.sin_addr.S_un.S_addr

This indicates the remote RTCP IP address as an integer where

Class A is byte 4

Class B is byte 3

Class C is byte 2

Class D is byte 1

Example: 10.128.100.100 would be 0x0A806464

args.destOptions.UDPTLopts.localAddr.sin_family

This indicates the type of local UDPTL addressing.

Valid values are:

TELE_CTRL_AF_INET_DEF

args.destOptions.UDPTLopts.localAddr.sin_port

This indicates the local UDPTL port number.

args.destOptions.UDPTLopts.localAddr.sin_addr.S_un.S_addr

This indicates the local UDPTL IP address as an integer where

Class A is byte 4

Class B is byte 3

Class C is byte 2

Class D is byte 1

Example: 10.128.100.100 would be 0x0A806464

args.destOptions.UDPTLopts.remoteAddr.sin_family

This indicates the type of remote UDPTL addressing.

Valid values are:

TELE_CTRL_AF_INET_DEF

args.destOptions.UDPTLopts.remoteAddr.sin_port

This indicates the remote UDPTL port number.

args.destOptions.UDPTLopts.remoteAddr.sin_addr.S_un.S_addr

This indicates the remote UDPTL IP address as an integer where

Class A is byte 4

Class B is byte 3

Class C is byte 2

Class D is byte 1

Example: 10.128.100.100 would be 0x0A806464

I - SR140 Security Capabilities

This appendix explains the SR140 security capabilities.

The security features introduced in SDK 6.8.0 require a security license to be installed on the system. The number of security channels must be equal or greater that the number of SR140 fax channels for security features to be enabled.

It has the following sections:

- *Secure RTP (SRTP)*
- *SIP over TLS*
- *FIPS*

Secure RTP (SRTP)

Data security protocols such as SRTP rely upon a separate key management system to securely establish encryption and/or authentication keys. The key exchange mechanism commonly used in VoIP sessions is called SDES (Security Descriptions for Media Streams), defined by *RFC 3711: "The Secure Real-time Transport Protocol (SRTP)"*. Using SDES the SRTP keys are negotiated in the SDP of the offer/answer model of a SIP exchange, using an SDP attribute called "crypto" which provides the cryptographic parameters of the requested media stream and other parameters that can be used to configure the SRTP media stream.

The use of SDES to exchange the keys is not a secure method, since the crypto key is transferred in the SDP as plain text string. The SDP "crypto" security description is normally used where IPsec, TLS, or some other encapsulating data-security protocol protects the SDP message. SIP layer security between the SIP Client and SIP Server, such as SIP TLS will be required by most customers to use SDES/SRTP.

If the fax session is re-invited to T.38, the T.38 media will not be secure. SRTP will only secure the fax media when G.711 Pass-through mode has been selected.

Example SRTP Configuration File

The following is an example SRTP configuration file.

```
srtp_accept = true
srtp_enforce = true
srtp_crypto_suite = AES_CM_128_HMAC_SHA1_80
srtp_master_key_len = 128
srtp_salting_key_len = 112
srtp_num_keys = 1
```

SIP over TLS

This section covers SIP over TLS. The SR140 supports up to TLS Version 1.2 as defined in RFC 5246.

SR140 behaves as a TLS server when receiving a SIP over TLS message and acts as a TLS client when placing a SIP over TLS call. The standard TLS connection setup is Client Hello (request), Server Hello with certificate (response), and then key exchange. Once completed, the SIP session will be performed using the exchanged keys. Below is a sample TLS message handshake flow for reference.

TLS Handshake

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data;
```

Configure Local Certificates

If the fax server application is acting as a TLS server, receiving incoming fax calls, it it must configure a local RSA certificate and/or DSS certificate. RSA certificate has an RSA key and DSS certificate has a DSA key. The TLS engine can hold two private key/certificate pairs: one RSA key/certificate and one DSS key/certificate. The certificate used depends on the cipher selected during TLS handshake.

To configure the use of an RSA certificate, the administrator must set ***local_rsa_private_key_filename*** and ***local_rsa_cert_filename*** in the TLS configuration file. These two filenames should contain the local certificate and key issued by a CA that identifies the local host name. If the key file specified by the ***local_rsa_private_key_filename*** is encrypted, ***local_rsa_private_key_password*** must be configured with the password to read the private key file. Otherwise ***local_rsa_private_key_password*** should be left as default value NULL.

To configure the use of a DSS certificate, the administrator must set ***local_dss_private_key_filename*** and ***local_dss_cert_filename*** in the TLS configuration file. These two filenames should contain the local certificate and key issued by a CA that identifies the local host name. If the key file specified by the ***local_dss_private_key_filename*** is encrypted, ***local_dss_private_key_password*** must be configured with the password to read the private key file. Otherwise ***local_dss_private_key_password*** should be left as default value NULL.

Configure Chain Certificates

An administrator may optionally configure ***chain_cert_number*** and ***chain_cert_filename*** if a local certificate is not issued by root CA, but by intermediate CAs. A maximum of 128 certificate files may be chained. The TLS configuration file may contain ***chain_cert_number*** copies of ***chain_cert_filename*** entries. Each entry is assigned a different certificate file name, which the SR140 reads into the ***chain_cert_filename*** array. Each ***chain_cert_filename*** must contain only one certificate in the chain.

The order of the chain certificates must start with the intermediate certificate that issues the local certificate and followed by one level up until reaching root CA certificate. For example, if *root.pem* signs *serverCA1.pem*, and *serverCA1.pem* signs *serverCA2.pem*, and *serverCA2.pem* signs *server.pem*, then the configuration should define the usage as the following:

```
chain_cert_number = 2
chain_cert_filename = serverCA2.pem
chain_cert_filename = serverCA1.pem
```

Note there is only one certificate chain in TLS engine, so that the same chain is appended to both types of certificates, RSA and DSS. An application cannot use different certificate chains for RSA and DSS certificates at the same time.

Configure CA Certificates

If the fax server application is acting as a TLS client, initiating outbound fax calls, ***ca_cert_number*** and ***ca_cert_filename*** must be configured in the TLS configuration file. These parameters should contain root CA certificate(s). More than one root CA certificates may be configured. The TLS configuration file may contain ***ca_cert_number*** copies of ***ca_cert_filename*** entries each assigned a different root CA certificate. The size maximum number of ***ca_cert_filename*** is 128. These may be required in mutual authentication where the TLS server wants to authenticate the TLS client, which is required to provide its own certificate.

Note that a fax application may act as a TLS server, TLS client, or both. If TLS is enabled with a valid TLS method, but neither server nor client configuration is provided, the SR140 will issue an error and start with the feature disabled.

Configure Certificate Revocation List (CRL)

A fax server application may optionally configure one or more CRL files by setting ***crl_number*** and ***crl_filename*** in the TLS configuration file. The TLS configuration file may contain ***crl_number*** copies of ***crl_filename*** entries, each assigned a different name of a PEM format file, up to a maximum of 128. When SR140 examines the incoming certificates, these CRLs will be consulted to decide whether the certificate has been revoked.

Example TLS configuration file

The following is an example TLS configuration file.

```
sip_tls_method=tlsv1.2
local_rsa_private_key_filename =
c:\brooktrout\tls\16110.key-cert.pem
local_rsa_cert_filename = c:\brooktrout\tls\16110.key-
cert.pem
ca_cert_number = 1
ca_cert_filename = c:\brooktrout\tls\rootcacert.pem
chain_cert_number = 1
chain_cert_filename = c:\brooktrout\tls\intercacert.pem
allow_self_signed_certs=true
```

TLS Cipher Suites

This appendix lists cipher suites supported by SR140 TLS ***local_cipher_suite*** string in SDK 6.8.0. The cipher suites are sorted by their strength (key size) for both SSLv3 and TLSv1.2.

Valid OpenSSL ciphers can be found at: *http://wiki.openssl.org/index.php/Manual:Ciphers(1)#SSL_v3.0_cipher_suites*

DEFAULT

The default cipher list. This is defined as ALL:!aNULL:!eNULL. This must be the first cipher string specified.

COMPLEMENTOFDEFAULT

The ciphers included in ALL, but not enabled by default. Currently, this is ADH and AECDH. Note that this rule does not cover eNULL, which is not included by ALL (use COMPLEMENTOFALL if necessary).

ALL

All cipher suites except the eNULL ciphers which must be explicitly enabled; the ALL cipher suites are reasonably ordered by default.

COMPLEMENTOFALL, !ALL

The cipher suites not enabled by ALL, currently being eNULL.

HIGH

"high" encryption cipher suites. This currently means those with key lengths larger than 128 bits, and some cipher suites with 128 bit keys.

MEDIUM

"medium" encryption cipher suites, currently some of those using 128 bit encryption.

LOW

"low" encryption cipher suites, currently those using 64 or 56 bit encryption algorithms but excluding export cipher suites.

EXP, EXPORT

Export encryption algorithms. Including 40 and 56 bits algorithms.

EXPORT40

40 bit export encryption algorithms.

EXPORT56

56 bit export encryption algorithms.

eNULL, NULL

The "NULL" ciphers that is those offering no encryption. Because these offer no encryption at all and are a security risk they are disabled unless explicitly included.

aNULL

The cipher suites offering no authentication. This is currently the anonymous DH algorithms and anonymous ECDH algorithms. These cipher suites are vulnerable to a "man in the middle" attack and so their use is normally discouraged.

kRSA, aRSA, RSA

Cipher suites using RSA key exchange, authentication or either respectively.

kDHr, kDHd, kDH

Cipher suites using DH key agreement and DH certificates signed by CAs with RSA and DSS keys or either respectively.

kDHE, kEDH

Cipher suites using ephemeral DH key agreement, including anonymous cipher suites.

DHE, EDH

Cipher suites using authenticated ephemeral DH key agreement.

ADH

Anonymous DH cipher suites, note that this does not include anonymous Elliptic Curve DH (ECDH) cipher suites.

DH

Cipher suites using DH, including anonymous DH, ephemeral DH and fixed DH.

kECDHr, kECDHe, kECDH

Cipher suites using fixed ECDH key agreement signed by CAs with RSA and ECDSA keys or either respectively.

kEECDH, kECDHE

Cipher suites using ephemeral ECDH key agreement, including anonymous cipher suites.

ECDHE, EECDH

Cipher suites using authenticated ephemeral ECDH key agreement.

AECDH

Anonymous Elliptic Curve Diffie Hellman cipher suites.

ECDH

Cipher suites using ECDH key exchange, including anonymous, ephemeral and fixed ECDH.

aDSS, DSS

Cipher suites using DSS authentication (i.e., the certificates carry DSS keys).

aDH

Cipher suites effectively using DH authentication (i.e., the certificates carry DH keys).

aECDH

Cipher suites effectively using ECDH authentication (i.e., the certificates carry ECDH keys).

aECDSA, ECDSA

Cipher suites using ECDSA authentication (i.e., the certificates carry ECDSA keys).

TLSv1.2, TLSv1, SSLv3

TLS v1.2, TLS v1.0 or SSL v3.0 cipher suites respectively. Note there are no cipher suites specific to TLS v1.1.

AES128, AES256, AES

Cipher suites using 128 bit AES, 256 bit AES, or either 128 or 256 bit AES.

AESGCM

AES in Galois Counter Mode (GCM): these cipher suites are only supported in TLS v1.2.

CAMELLIA128, CAMELLIA256, CAMELLIA

Cipher suites using 128 bit CAMELLIA, 256 bit CAMELLIA or either 128 or 256 bit CAMELLIA.

3DES

Cipher suites using triple DES.

DES

Cipher suites using DES (not triple DES).

RC4

Cipher suites using RC4.

RC2

Cipher suites using RC2.

IDEA

Cipher suites using IDEA.

SEED

Cipher suites using SEED.

MD5

Cipher suites using MD5.

SHA1, SHA

Cipher suites using SHA1.

SHA256, SHA384

Cipher suites using SHA256 or SHA384.

PSK

Cipher suites using pre-shared keys (PSK).

SUITEB128, SUITEB128ONLY, SUITEB192

Enables suite B mode operation using 128 (permitting 192 bit mode by peer) 128 bit (not permitting 192 bit by peer) or 192 bit level of security respectively. If used, these cipher strings should appear first in the cipher list and anything after them is ignored. Setting Suite B mode has additional consequences required to comply with RFC 6460. In particular, the supported signature algorithms is reduced to support only ECDSA and SHA256 or SHA384, only the elliptic curves P-256 and P-384 can be used and only the two suite B compliant cipher suites (ECDHE-ECDSA-AES128-GCM-SHA256 and ECDHE-ECDSA-AES256-GCM-SHA384) are permissible.

FIPS

FIPS is the Federal Information Processing Standards. See *http://www.itl.nist.gov/fipspubs* for more information.

The FIPS Object Module was designed and implemented to meet FIPS 140-2, Level 1 requirements. The FIPS Object Module provides confidentiality, integrity signing, and verification services.

Master Index

Numerics

1536K calls
  L4L3mDISABLE_B_CHANNEL 5: 1009, 5: 1013, 5: 1026, 5: 1074, 5: 1079
1TR6 German variant 5: 1039, 6: 1213
384K calls
  L4L3mDISABLE_B_CHANNEL 5: 1009, 5: 1013, 5: 1026, 5: 1074, 5: 1079
4ESS switch
  L4L3mENABLE_PROTOCOL 5: 1038, 5: 1051
5ESS switch
  L4L3mENABLE_PROTOCOL 5: 1038, 5: 1051

A

Accepting incoming call 2: 280
Address structure 6: 1343, 6: 1484
Address, checking for 1: 50
Alerting and Connecting Data Message common structure 5: 851
ALERTING Q.931
  L3L4mCONNECT 5: 1102
  L4L3mALERTING_REQUEST 5: 1006
Alerts
  handling in Millennial 1: 92
Analog DID
  line characteristics 6: 1195
  port configuration 6: 1194
Analog Loop Start
  port configuration 6: 1197
ANI on Demand 5: 1057, 5: 1058, 5: 1088
Answering Call function
  BfvCallWaitForAccept 2: 331
Answering incoming call 2: 280, 2: 331
AOC structure information macro 1: 84
API debug mode 1: 261
  directing output 1: 234
  enabling 1: 237, 1: 239
API_V1 macro 1: 87
API_V2 macro 1: 87
API_V3 macro 1: 87
API_V4 macro 1: 87
API_VER_NUM macro 1: 86
API_VERSION macro 1: 87
Application behavior information macro 1: 84
Argentina R2 parameter file 6: 1459
args_cc structure 2: 439–2: 466
args_telephone structure 2: 468–2: 480
Arguments macro 1: 84
ASCII strip infopkt 6: 1404
AT&T
  4ESS Fast Connect Feature
    L3L4mALERTING 5: 1087
    L3L4mCONNECT 5: 1102
    L3L4mPROGRESS 5: 1118
    L3L4mSETUP_IND 5: 1133
    L4L3mALERTING_REQUEST 5: 1007
    L4L3mPROGRESS_REQUEST 5: 1067
  4ESS switch
    L4L3mENABLE_PROTOCOL 5: 1038, 5: 1051
  5ESS switch
    L4L3mENABLE_PROTOCOL 5: 1038, 5: 1051
  Accunet service 5: 1014, 5: 1130
  ANI on Demand feature 5: 1089
  International 800 service 5: 1014, 5: 1131
  Megacom service 5: 1130
  Software Defined Network (SDN) 5: 1130
  Variabill feature 5: 1058, 5: 1093

AT&T Custom variant 5: 1051
Australia, dialing requirements 6: 1450
Automatic Gain Control (AGC) 6: 1154
  enabling 3: 577, 3: 587, 3: 594

B

Basic Rate Interface (BRI) 6: 1201
B-channel
  disabling 5: 1028
  idling 5: 1073
  overriding call type setting 5: 1016
B-channel maintenance
  L3L4mRESTART 5: 1142
  L4L3mDISABLE_B_CHANNEL 5: 1141
  L4L3mRESTART 5: 1141
  Q.931 messages 5: 1140
B-channel negotiation
  L4L3mALERTING_REQUEST 5: 1007
  L4L3mCALL_PROCEEDING_REQUEST
    5: 1008, 5: 1010
  L4L3mENABLE_PROTOCOL 5: 852
  L4L3mSETUP_ACK_REQUEST 5: 1009, 5: 1015,
    5: 1026, 5: 1031, 5: 1074, 5: 1079
Bearer capability data 5: 1016
Beginning of page infopkt 6: 1417
BFAX_V2 macro 1: 89
Bfv API libraries 1: 30
BfvBoardNotify 1: 171, 1: 190
BfvBoardStateGet 1: 178
BfvBoardStateSet 1: 180
BfvBoardTemperatureGet 1: 182
BfvBoardTemperatureThreshSet 1: 184
BfvBoardTest 1: 186, 1: 195, 1: 198, 1: 206
BfvCallAccept 2: 280
BfvCallCtrlClose 2: 283, 2: 291
BfvCallCtrlInit 2: 284
BfvCallDisconnect 2: 287
BfvCallHold 2: 289
BfvCallReconfigureHostModule 2: 291
BfvCallReject 2: 293
BfvCallRetrieve 2: 295
BfvCallRingDetect 2: 297
BfvCallSendAlerting 2: 301
BfvCallSetup 2: 303
BfvCallSignalingStateMonitor 2: 319
BfvCallSignalingStateSet 2: 323
BfvCallStatus 2: 326
BfvCallSWClearConns 1: 119
BfvCallSWConnect 1: 121, 1: 127
BfvCallSWGetConns 1: 142
BfvCallSWGetInfo 1: 146
BfvCallTransferComplete 2: 329
BfvCallWaitForAccept 2: 331
BfvCallWaitForAlerting 2: 334
BfvCallWaitForComplete 2: 337
BfvCallWaitForDivert 2: 345
BfvCallWaitForHold 2: 345
BfvCallWaitForRelease 2: 347
BfvCallWaitForRetrieve 2: 350
BfvCallWaitForSetup 2: 352
BfvCallWaitTransferComplete 2: 359
BfvCheckAddress 1: 50
BfvCheckFacility 1: 52
BfvCPGen 3: 492
BfvCPGenAdv 3: 494
BfvDataCP 3: 498
BfvDataFSK 4: 634
BfvDebugFuncSet 1: 234
BfvDebugInitData 1: 236
BfvDebugModeSet 1: 237
BfvDebugModeSetAdv 1: 239
BfvDialDBCheck 2: 423
BfvDialDBList 2: 425
BfvDialDBUpdate 2: 428
BfvErrorMessage 1: 244
BfvFaxAbort 4: 638
BfvFaxBegin 4: 640
BfvFaxBeginRaw 4: 647
BfvFaxBeginReceive 4: 654
BfvFaxBeginSend 4: 659
BfvFaxBeginSendRaw 4: 662
BfvFaxBeginSendTiff 4: 667
BfvFaxBeginTiff 4: 670
BfvFaxDownloadFont 4: 675
BfvFaxEndOfDocument 4: 682
BfvFaxEndReception 4: 684
BfvFaxGetRemoteInfo 4: 686
BfvFaxHeader 4: 688
BfvFaxNextPage 4: 692
BfvFaxNextPageDCX 4: 695
BfvFaxNextPageRaw 4: 698
BfvFaxNextPageTiff 4: 701
BfvFaxPageParams 4: 704
BfvFaxPoll 4: 706

BfvFaxRcvPageDCX 4: 711
BfvFaxRcvPageTiff 4: 713
BfvFaxReceive 4: 716
BfvFaxReceiveData 4: 721
BfvFaxReceiveFile 4: 725
BfvFaxReceivePage 4: 728
BfvFaxReceivePages 4: 730
BfvFaxSend 4: 732
BfvFaxSendData 4: 736
BfvFaxSendFile 4: 738
BfvFaxSendPage 4: 741
BfvFaxSendPageDCX 4: 744
BfvFaxSendPageTiff 4: 746
BfvFaxSetLocalId 4: 748
BfvFaxSetNSF 4: 750
BfvFaxSetReceiveFmt 4: 753
BfvFaxSetSubPwdSep 4: 756
BfvFaxStripParams 4: 759
BfvFaxT30Holdup 4: 763
BfvFaxT30Params 4: 769
BfvFaxWaitForTraining 4: 776
BfvFeatureSetDownload 1: 95
BfvFeatureSetDownloadData 1: 97
BfvFeatureSetQuery 1: 99
BfvFirmwareDownload 1: 101
BfvFirmwareDownloadData 1: 105
BfvGetVar 1: 212
BfvHistoryClear 1: 246
BfvHistoryClearModChan 1: 248
BfvHistoryClearUnit 1: 250
BfvHistoryDump 1: 252
BfvHistoryDumpModChan 1: 255
BfvHistoryDumpUnit 1: 258
BfvInfoPktClose 3: 601
BfvInfopktFseek 3: 603
BfvInfopktFtell 3: 605
BfvInfopktGet 3: 607
BfvInfopktOpen 3: 609
BfvInfopktOpenMem 3: 611
BfvInfopktPut 3: 615
BfvInfopktUnget 3: 617
BfvInfopktUser 3: 619
BfvLineAlert 1: 215
BfvLineAnswer 2: 362
BfvLineAttach 1: 54
BfvLineCallProgressDisable 3: 502
BfvLineCallProgressEnable 3: 504
BfvLineCallProgressProgram 3: 509
BfvLineCCProtocolGet 2: 365
BfvLineConfig 1: 57
BfvLineDetach 1: 60
BfvLineDialString 2: 368
BfvLineDumpStructure 1: 261
BfvLineInfo 1: 61
BfvLineOrigCallDB 2: 431
BfvLineOriginateCall 2: 374
BfvLineReset 1: 63
BfvLinesAvail 1: 67
BfvLineTerminateCall 2: 392
BfvLineTransfer 2: 396
BfvLineTransferCancel 2: 402
BfvLineTransferCapabilityQuery 2: 404
BfvLineTransferComplete 2: 406
BfvLineWaitForCall 2: 408
BfvLoopCurrentDetectDisable 2: 416
BfvLoopCurrentDetectEnable 2: 418
BfvMemAllocFuncsSet 1: 218
BfvModuleConfigSpecsGet 1: 109
BfvModuleDeactivate 1: 69
BfvModuleInfo 1: 71
BfvNetworkConfigGet 1: 149
BfvNetworkConfigSet 1: 153
BfvNetworkQuery 1: 158
BfvPromptClose 3: 622
BfvPromptOpen 3: 624
BfvPromptPlay 3: 531
BfvRcvProcessPkt 1: 221
BfvSessionAttach 1: 78
BfvSessionDetach 1: 81
BfvSetSingleVar 1: 224
BfvSpeechEchoCancelControl 3: 535, 3: 541
BfvSpeechModify 3: 538
BfvSpeechPlay 3: 541
BfvSpeechPlayData 3: 544
BfvSpeechPlayFile 3: 551
BfvSpeechPlayWave 3: 557
BfvSpeechQuerySummationGroup 3: 541
BfvSpeechRecord 3: 561
BfvSpeechRecordData 3: 570
BfvSpeechRecordFile 3: 580
BfvSpeechRecordWave 3: 589
BfvTelephGetInfo 1: 163
BfvTelephReset 1: 165
BfvTelephSave 1: 167
BfvTiffClose 4: 783
BfvTiffOpen 4: 785

BfvTiffReadIFD 4: 787
BfvTiffReadImage 4: 790
BfvTiffReadRes 4: 792
BfvTiffWriteIFD 4: 794
BfvTiffWriteImage 4: 797
BfvTiffWriteRes 4: 799
BfvToneDetectDisable 3: 513
BfvToneDetectEnable 3: 515
BfvToneFlush 3: 518
BfvToneGet 3: 519
BfvTonePeek 3: 521
BfvTonePlay 3: 523
BfvTonePlayBeep 3: 525
BfvToneUnget 3: 528
Billing rate 5: 1093
Blacklisted telephone numbers 2: 429
Board configuration 1: 57
Board notify functions 1: 171–1: 189, 1: 190–??
Board state monitoring functions 1: 69, 1: 178
Boot ROM firmware 1: 111
BOSTON Host Service 2: 286
BOSTON Simple Message Interface, see BMSI 5: 804
Brazil R2 parameter file 6: 1462
BRI protocol port configuration 6: 1201
BRI Protocol Stack, initializing 5: 830
BROOKTROUT_MILLENNIAL macro 1: 86
BSMI
  call reference value 5: 847
  common header structure 5: 832
  directories and files 5: 806
  error return values 5: 824
  L4 reference value 5: 846
  message sequence examples 5: 827
  message structure 5: 831
  message types 5: 833
  R2 signaling messages 5: 877
BsmiCloseAdapter 5: 809
BsmiControlRead 5: 810
BsmiControlWrite 5: 812
BsmiLineAlert 5: 813
BsmiModuleList 5: 815
BsmiOpenAdapter 5: 817
BsmiResetAdapter 5: 819
BT_API_SET_VER macro 1: 86
BT_ARGS macro 1: 84
BT_BIG_ENDIAN macro 1: 85
BT_CBARGS macro 1: 85
BT_CPARM.CFG file 1: 117, 6: 1154, 6: 1439, 6: 1441
BT_LITTLE_ENDIAN macro 1: 85
BT_ZERO macro 1: 82
btcall.cfg file 1: 117, 2: 279, 6: 1152
  user-defined configuration file 1: 65
  user-defined parameters 6: 1152
BTLINE structure 1: 47
  dumping 1: 261
Build number
  macros 1: 87
Bus configuration
  H.1xx clocking 6: 1188
BYTE_SWAP_LONG macro 1: 85
BYTE_SWAP_SHORT macro 1: 85

C

Call alerting
  function 2: 301
Call clearing
  reporting 5: 1099, 5: 1100
Call control
  answering 2: 280
  configuring environment 2: 284
  disabling ISDN signaling 2: 283, 2: 291
  initializing 2: 284
  shutting library 2: 283, 2: 291
Call control configuration
  analog DID port 6: 1194
  analog loop start port 6: 1197
  BRI port 6: 1201
  E1 CAS port 6: 1217
  E1 CAS R2 port 6: 1221
  E1 ISDN port 6: 1209
  E1 QSIG port 6: 1223
  Ethernet interface 6: 1286–6: 1290
  file examples 6: 1296–6: 1318
  file format 6: 1171
  global module 6: 1178
  H.1xx clocking 6: 1188
  Internet Protocol (IP) 6: 1245–6: 1291
  JATE redial restriction 6: 1181
  modifying file 6: 1170
  parameters 6: 1171–6: 1291
  T1 ISDN port 6: 1232
  T1 RBS port 6: 1240

trace options 6: 1176
Call Control functions
high-level data structure 2: 468–2: 480
high-level summary 2: 273
ISDN services 2: 277
low-level data structure 2: 439–2: 466
low-level summary 2: 274
protocol-specific 2: 276
Call control messages summary, BSMI 5: 836
Call Control runtime library 2: 272
Call deflection 5: 1053
Call Disconnect function
BfvCallDisconnect 2: 287
BfvCallWaitForRelease 2: 347
Call flows
R2 signaling 5: 910
Call ID common structure 5: 854
Call placement codes 6: 1373
CALL PROCEEDING Q.931 message 5: 1042
L3L4mSETUP_IND 5: 1133
L4L3mCALL_PROCEEDING_REQUEST 5: 1011
L4L3mENABLE_PROTOCOL 5: 1042
Call progress
adapting to international signal specifications 6: 1420
ANSWER call mode 6: 1423
bandpass filter 6: 1419
board-determined results 6: 1423
cadence detection algorithm 6: 1419
call progress buffer 6: 1423
call protocol code 6: 1425
fax mode 6: 1425
raw mode 6: 1425
voice mode 6: 1425
detecting human 6: 1422
disabling 3: 502
discrete filters 6: 1419
enabling 3: 504
initiating call progress 6: 1423
BfvLineCallProgressEnable 6: 1423
BfvLineOriginateCall 6: 1424
modes 3: 504
ORIGINATE call mode 6: 1423
processing signals 6: 1419
programs frequency 3: 509
reporting 2: 374, 2: 396
reporting results 6: 1421
final 6: 1421
HUMAN 6: 1422
intermediate 6: 1421
sending CNG 6: 1426
setting the call progress mode 6: 1425
signal interpretation 2: 388, 3: 508
special information tones (S.I.T.) 6: 1434
using special features 6: 1426
when dialing 6: 1426
Call progress results
board-determined 6: 1423
custom 6: 1436
custom templates 6: 1436
dialing 2: 374, 2: 396
final 6: 1438
transferring a call 2: 396
Call progress signals 6: 1427–6: 1433
ANSWER 6: 1427
ANSWER_TONE_DETECT 6: 1427
BUSY1 6: 1427
BUSY2 6: 1428
CNGDETCT 6: 1428
CONFIRM 6: 1428
DIALTON 6: 1429
G2DETCT 6: 1429
HUMAN 6: 1429
PULSE 6: 1430
QUIET 6: 1431
RECALL 6: 1431
RING1 6: 1431
RMTOFFHK 6: 1432
RNGNOANS 6: 1432
ROBUSY 6: 1432
SILENCE 6: 1433
SPECIALCP 6: 1433
Call reference value, BSMI 5: 847
Call state, retrieving 2: 326
Call status
diversion 6: 1353
redirection 6: 1353
Call switching functions 1: 119–1: 148
Call Tracer Utility 1: 231
Call transfer
completing 2: 329, 2: 406
disabling 6: 1200, 6: 1208, 6: 1215, 6: 1220, 6: 1231, 6: 1239, 6: 1243
function 2: 289, 2: 295, 2: 329, 2: 345, 2: 350, 2: 359, 2: 396–2: 407
setup 2: 303

transfer_variant values 6: 1200, 6: 1208, 6: 1215, 6: 1220, 6: 1231, 6: 1239, 6: 1243
CALL_RES parameters 6: 1352
callctrl.cfg file 2: 272, 2: 279, 6: 1170
Called Party
CCITT number types 5: 856
common structure 5: 855
Number IE 5: 855
Caller ID 6: 1352, 6: 1354
Caller name 6: 1352, 6: 1354
Calling Party
AT&T numbering plan 5: 872
CCITT number types 5: 858, 5: 872
CCITT numbering plan 5: 858
common structure 5: 857
Number IE 5: 857
Calls
accepting incoming 2: 280
canceling call transfer 2: 402
complete answering 2: 331
complete outgoing 2: 334, 2: 337
completing
call transfer connection 2: 406
completing call transfer 2: 359
completing transfer 2: 329
disconnecting 2: 392
divert incoming 2: 345
finish answering 2: 331
finish diverting 2: 345
finish outbound 2: 334, 2: 337
finish transition from hold state 2: 350
making enquiry 2: 303
monitoring signaling state 2: 319
outgoing 2: 303
placing in hold state 2: 289, 2: 345
rejecting incoming call 2: 293
sending alerting message 2: 301
setting signaling state 2: 323
starting enquiry 2: 303
transferring 2: 303
transition out of hold state 2: 295, 2: 350
turning off loop current detection 2: 416
turning on loop current detection 2: 418
waiting for incoming 2: 408
waiting to detect 2: 352
waiting to detect incoming 2: 408
Canada, dialing requirements 6: 1451
Cancel call transfer 2: 402
Capability, channel transfer 2: 404
CAS protocols 5: 934–5: 1004
configuring port 6: 1217–6: 1222
Cause Data common structure 5: 860
Cause IE values 6: 1386
CED wait time 6: 1154
Channel statistics 5: 1071, 5: 1115
Channel-Associated Signaling (CAS) protocols 5: 934–5: 1004
configuring port 6: 1217–6: 1222
Channels
available 1: 67
closing 1: 60, 1: 81
finish answering process 2: 331
finish diverting process 2: 345
finish outbound process 2: 334, 2: 337
initializing 1: 65
interrupting active 1: 215
opening 1: 54, 1: 78
resetting 1: 63
retrieving board type 1: 61
retrieving call state 2: 326
retrieving I/O port address 1: 61
transfer capability 2: 404
Checking for
address 1: 50
facility 1: 52
Checksum macro 1: 115
China R2 parameter file 6: 1469
Clear information macro 1: 82
Clearing connected call 2: 287, 2: 347
Clocking
H.1xx configuration 6: 1188
Closing call control library 2: 283, 2: 291
Codeset shifts for IEs 5: 865
Coding standards 5: 868, 5: 1063
Command lines
parsing options 1: 228
Comment
control processor firmware 1: 113
ROM firmware 1: 111
Common structures, BSMI
IISDN_AL_CON_DATA 5: 851
IISDN_CALL_ID 5: 854
IISDN_CALLED_PARTY 5: 855
IISDN_CALLING_PARTY 5: 857
IISDN_CAUSE 5: 860
IISDN_CONNECTED_ADDRESS 5: 863

IISDN_IE_STRUCT 5: 865
IISDN_PROGRESS_IND 5: 867
IISDN_Q922_DLCI 5: 870
IISDN_REDIRECT_NUM 5: 871
IISDN_USER_INFO 5: 875
Completing
call answer 2: 331
call diversion 2: 345
call transfer 2: 329, 2: 359
call transfer connection 2: 406
outbound call 2: 334, 2: 337
Compression types for fax transmission 6: 1164
Configuration files 1: 117
btcall.cfg 6: 1152
call control 2: 279, 6: 1170
call control examples 6: 1296–6: 1318
callctrl.cfg 2: 272, 6: 1170
ecc.cfg 2: 272, 6: 1150
teleph.cfg 2: 272, 6: 1150
user-defined 6: 1152
Configuring call control 2: 284, 6: 1170
Configuring Ethernet interface 6: 1286–6: 1290
Configuring IP call control 6: 1245–6: 1291
Configuring IP call control stack 6: 1291
Connected Address
CCITT number types 5: 864
CCITT numbering plan 5: 864
common structure 5: 863
Connecting call transfer 2: 406
Control processor firmware macro 1: 112
Country codes parameter 6: 1155
Country telephone parameter file
BT_CPARM.CFG file 6: 1154, 6: 1441
Country-specific parameters file 1: 117, 6: 1154, 6: 1441
CPU type macro 1: 83
Current, loop
turning off detection 2: 416
turning on detection 2: 418
Custom call progress results
CUSTOM_DIS_CAD0 6: 1437
CUSTOM_DIS_FREQ0 6: 1437
CUSTOM_FREQ0 6: 1436, 6: 1437
Customizing IP call control stack 6: 1257
Czech Republic, dialing requirements 6: 1451

D

Data Link Connection Identifier (DCLI) 5: 849
D-channel
determining Layer 2 status 5: 1072
Primary NFAS D-channel 5: 1124
reporting status 5: 1119
D-channel maintenance 5: 1147
L4L3mDIABLE_PROTOCOL 5: 1147
L4L3mENABLE_PROTOCOL 5: 1147
DCX files
receiving a page 4: 711
sending end-of-page 4: 695
transmitting PCX pages 4: 744
Debug functions 1: 234–1: 238
Debug mode 6: 1156
Debug output, enable 2: 278
Debugging tools 1: 261
API debug mode 1: 234, 1: 237, 1: 239
Call Tracer utility 1: 231
Decoded DCS, DIS, DTC info structures 6: 1367, 6: 1493
Denmark, dialing requirements 6: 1452
Destination Millennial address macro 1: 90
Detecting
incoming call 2: 352
ring 2: 297
Detecting loop current 2: 416, 2: 418
Dialing database
blacklisting numbers 2: 429
checking 2: 423
country restrictions 2: 426
reading contents 2: 425
updating/reading 2: 428
using functions 6: 1448
Dialing mode 6: 1167
Dialing phone numbers 2: 374
Direct Inward Dialing (DID)
port configuration 6: 1194
DISCONNECT Q.931 message 5: 1104
Disconnecting call 2: 287, 2: 347, 2: 392
Diverting Call function
BfvCallWaitForDivert 2: 345
Diverting incoming call 2: 345
DLCI
defined 5: 849
format 5: 849
macro example 5: 849

negotiation 5: 870
DLL replacing
BT_API_SET_VER 1: 86
dll_... functions 1: 211
DMS-100 switch 5: 1051
DMS-250 switch 5: 1051, 5: 1053
Document parameters infopkt 6: 1406
Downloading firmware 1: 101, 1: 105
DTMF
configuration parameters 6: 1157–6: 1160
disabling detection 3: 513
enabling detection 3: 515
parameters, configuration 6: 1157–6: 1160
tone dialing mode 6: 1167
Dump History Utility 1: 252

E

E1 CAS port configuration 6: 1217
E1 CAS R2 port configuration 6: 1221
E1 QSIG port configuration 6: 1223
ecc.cfg file 2: 272, 6: 1150
Echo Cancellation 3: 535, 3: 541
ECM 6: 1160
Emergency number, JATE 6: 1182
Ending call 2: 287, 2: 347
End-of-page
sending 4: 692
sending for DCX files 4: 695
sending for noninfopkt files 4: 698
sending for TIFF-F files 4: 704
End-of-speech parameter infopkt 6: 1400
Enhanced fax format page infopkt 6: 1408
Enquiry call, placing 2: 303
Ericsson MD-110 switch
switch type 5: 1039
variants supported 5: 1051
Error
BSMI return values 5: 824
correction mode (ECM) 6: 1160, 6: 1161
detection 6: 1338
interrupt macro 1: 263, 1: 264
L3L4m messages defined 5: 1106
message strings 1: 244
multiplication value 6: 1161
reporting codes 5: 1106
threshold values 6: 1162
Error handling functions 1: 244–1: 262
Ethernet interface
configuration parameters 6: 1286–6: 1290
European Community (EC), dialing requirements 6: 1452
Event messages
debug output 2: 278
Examples
BSMI message sequence 5: 827
call control configuration files 6: 1296–6: 1318

F

Facilities Data Link (FDL)
passing LAP-D messages 5: 1037
Facility checking 1: 52
FACILITY Q.931
L4L3mFACILITY_REQUEST 5: 1052
L4L3mFEATURE_REQUEST 5: 1057
Fax
aborting transmission/reception 4: 638
completing reception 4: 684
dialing, sending, receiving, answering 4: 706
dumping history 1: 252, 1: 258
enabling T.30 holdup 4: 763, 4: 769
formatting headers/footers 4: 688
initiating 4: 640, 4: 654
preparing line data structure 4: 659
providing turnaround polling 4: 776
receive DIS, DTS, or DCS 4: 686
receiving a TIFF-F file 4: 713
receiving based on infopkt stream 4: 716
receiving multiple pages 4: 730
receiving remote ID 4: 686
reporting training_complete 4: 776
sending documents based on infopkt stream 4: 732
sending end-of-page 4: 692
sending end-of-page command 4: 682
setting format of received data 4: 753
setting FSK messages 4: 750, 4: 756
setting local IDs 4: 748
setting T.30 parameters 4: 769
switching to voice mode 4: 675
T.38 configuration 6: 1248
transferring G3 pages to infopkt stream 4: 728
transmitting entire page 4: 741

Fax header parameters infopkt 6: 1409
FAX_RES parameters 6: 1362
Feature set functions 1: 95–1: 100
Finish
  answering call 2: 331
  call disconnection 2: 347
  diverting call 2: 345
  outbound call 2: 334, 2: 337
Firmware
  downloaded macro 1: 115
  downloading 1: 101, 1: 105
  ID macro 1: 115
  information macros 4: 779
  type macro 1: 115
Flexible billing 5: 1058
Font files
  downloading parameters 6: 1163
Fonts
  Dialogic supported 4: 677
  downloading 4: 675
  information macro 4: 779
Frame Relay
  common structures 5: 1008
  DLCI negotiation 5: 870
France, dialing requirements 6: 1452
FSK
  data 4: 634
  information macros 4: 778
  setting up messages 4: 750, 4: 756
  signal definitions 4: 635
Function pointer arguments macro 1: 85
Functions
  locator directory 1: 34–1: 45

G

Germany, dialing requirements 6: 1453
getopt 1: 228
Global Unique IDentifier (GUID) 6: 1288

H

H.1xx clocking configuration 6: 1188
H.323 IP call control parameters 6: 1258
H0 calls
  L4L3mCALL_REQUEST 5: 1015
  L4L3mDISABLE_B_CHANNEL 5: 1009, 5: 1013, 5: 1026, 5: 1074, 5: 1079
H11 calls
  L4L3mCALL_REQUEST 5: 1016
  L4L3mDISABLE_B_CHANNEL 5: 1009, 5: 1013, 5: 1026, 5: 1074, 5: 1079
Handling alerts
  Millennial 1: 92
Hangup codes 6: 1372–6: 1385
  API created codes 6: 1385
  call placement codes 6: 1373
  miscellaneous 6: 1384
  phase C codes 6: 1383
  receive phase B codes 6: 1380
  receive phase D codes 6: 1382
  transmit phase A codes 6: 1373
  transmit phase B codes 6: 1374
  transmit phase D codes 6: 1376
HDLC channels
  disabling protocol stack 5: 1029
Headers/footers, setting up 4: 688
High-level call control function
  BfvCallReject 2: 293
  BfvLineAnswer 2: 362
  BfvLineCCProtocolGet 2: 365
  BfvLineDialString 2: 368
  BfvLineOriginateCall 2: 374
  BfvLineTerminateCall 2: 392
  BfvLineTransfer 2: 396
  BfvLineTransferCancel 2: 402
  BfvLineTransferCapabilityQuery 2: 404
  BfvLineTransferComplete 2: 406
  BfvLineWaitForCall 2: 408
  BfvLoopCurrentDetectDisable 2: 416
  BfvLoopCurrentDetectEnable 2: 418
History
  clearing 1: 246, 1: 248, 1: 250
  dumping 1: 252, 1: 258
  functions 1: 246–1: 260
Hold state 2: 289, 2: 295, 2: 345, 2: 350
Hong Kong, dialing requirements 6: 1453
Hookflash support, BSMI 5: 963
Host module
  reconfiguring call control 2: 291

I

IFD entries

interpreting 4: 792
reading in TIFF-F files 4: 787
writing into TIFF-F files 4: 794
IISDN_ABCD_DATA
L3L4mABCD_SIGNAL_DATA 5: 970
L4L3mREQ_ABCD_DATA 5: 957
IISDN_ABCD_SIGNALS
L3L4mABCD_SIGNAL_DATA 5: 970
L3L4mALERTING 5: 913, 5: 973
L4L3mREQ_ABCD_DATA 5: 957
IISDN_AL_CON_DATA 5: 851
L3L4mALERTING 5: 913, 5: 973, 5: 1087
L3L4mCONNECT 5: 924, 5: 984, 5: 1102
L4L3mALERTING_REQUEST 5: 1006
L4L3mCONNECT_REQUEST 5: 943, 5: 1023
L4L3mFORCE_CONNECTION_REQUEST 5: 955
IISDN_ANI_DATA
L3L4mANI 5: 1088
IISDN_BCHANNEL_ID
L4L3mDISABLE_B_CHANNEL 5: 1025
L4L3mENABLE_B_CHANNEL 5: 1030
L4L3mENABLE_CAS 5: 951
L4L3mRESTART 5: 1073
IISDN_BOARD_ID
L3L4mBOARD_ID 5: 1094
IISDN_CALL_ID 5: 854
IISDN_CALL_PROC_DATA
L3L4mCALL_PROCEEDING 5: 1096
L4L3mCALL_PROCEEDING_REQUEST 5: 1008
IISDN_CALL_REQ_DATA
L4L3mCALL_REQUEST 5: 885, 5: 937, 5: 1012
IISDN_CALLED_PARTY 5: 855
L3L4mSETUP_IND 5: 930, 5: 1001
L4L3mCALL_REQUEST 5: 936
IISDN_CALLER_ID_DATA
L3L4mCALLER_ID_DETECTED 5: 975
IISDN_CALLING_PARTY 5: 857
L3L4mSETUP_IND 5: 930, 5: 1001
L4L3mCALL_REQUEST 5: 936
IISDN_CAS_SIGNALING_BITS
L3L4mCAS_SIGNALING_BIT_STATUS 5: 917, 5: 977
IISDN_CAS_SIGNALING_BITS_DATA
L3L4mCAS_SIGNALING_BIT_STATUS 5: 917, 5: 977
IISDN_CAUSE 5: 860
L3L4mCLEAR_REQUEST 5: 921, 5: 980
L3L4mDISCONNECT 5: 926, 5: 986
IISDN_CLR_DATA
L3L4mCLEAR_REQUEST 5: 921, 5: 980, 5: 1098
L3L4mCLEAR_WITH_RESTART_REQUEST 5: 1100
L3L4mDISCONNECT 5: 926, 5: 986, 5: 1104
L4L3mCLEAR_REQUEST 5: 1020
IISDN_CONFIG_DATA
L3L4mCONFIGURATION_STATUS 5: 982
L4L3mREQ_ABCD_DATA 5: 957
L4L3mREQ_CONFIGURATION 5: 958
L4L3mSET_CONFIGURATION 5: 960
IISDN_CONNECTED_ADDRESS 5: 863
IISDN_D_CHAN_STAT
L3L4mPROTOCOL_STATUS 5: 1119
IISDN_DIAL_DATA
L4L3mDIAL 5: 945
IISDN_DOWNLOAD_DATA
L3L4mACK_DOWNLOAD 5: 971
IISDN_E1_CAS_R2_DATA
L4L3mENABLE_CAS 5: 951
IISDN_ENA_PROTO_DATA
L4L3mENABLE_PROTOCOL 5: 1033
IISDN_FACILITY_DATA
L4L3mFACILITY_REQUEST 5: 1052
IISDN_HARDWARE_DATA
L4L3mSET_HARDWARE 5: 1077
IISDN_IE_STRUCT 5: 865
IISDN_INFO_DATA
L3L4mINFO_REQUEST 5: 1111
L4L3mINFO_REQUEST 5: 1059
IISDN_JATE_REDIAL
L4L3mJATE_REDIAL 5: 1061
IISDN_l1mode values 5: 1077
IISDN_L2_STATS
L3L4mL2_STATS 5: 1113
IISDN_LEVEL3_CNFG
L3L4mRAW_QDATA 5: 1126
IISDN_LINE_STATUS
L3L4mLINE_STATUS 5: 1116
IISDN_PROGRESS
L3L4mPROGRESS 5: 996, 5: 1118
L4L3mPROGRESS_REQUEST 5: 1063
IISDN_PROGRESS_IND 5: 867
IISDN_Q922_DLCI 5: 870
IISDN_Q931_CNFG
L3L4mCONN_ACT_IND 5: 1103
IISDN_R2_CALL_STATUS

L3L4mSTATUS_IND 5: 1002
IISDN_R2_INTERREGISTER_PARAMS structure 5: 901
IISDN_RAW_QDATA
L3L4mRAW_DATA 5: 1126
IISDN_REDIRECT_NUM 5: 871
IISDN_RING_STATUS_DATA
L3L4mRING_STATUS 5: 997
IISDN_ROBBED_BIT_DATA
L4L3mCLEAR_REQUEST 5: 939
L4L3mENABLE_CAS 5: 951
IISDN_SETUP_ACK
L4L3mSETUP_ACK_REQUEST 5: 1078
IISDN_SETUP_DATA
L3L4mSETUP_IND 5: 930, 5: 1001, 5: 1129
IISDN_SIGNAL_DURATION_DATA
L4L3mTX_HOOKFLASH 5: 963
L4L3mTX_WINK 5: 965
IISDN_UNIVERSAL
L3L4mUNIVERSAL 5: 1135
L4L3mUNIVERSAL 5: 1081
IISDN_UPLOAD_DATA
L3L4mACK_UPLOAD 5: 972
IISDN_USER_INFO 5: 875
L3L4mUSER_INFO 5: 1137
L4L3mUSER_INFO 5: 1085
IISDN1mod values 5: 1035
IISDNacd values 5: 1123
IISDNani values 5: 1088
IISDNbcs values 5: 898, 5: 901, 5: 919, 5: 948, 5: 951, 5: 979, 5: 1090, 5: 1092
IISDNbs values 5: 1132
IISDNcalltyp values 5: 1040, 5: 1129
IISDNcfgtype values 5: 974
IISDNcod values 5: 1063
5: 861, 5: 868
IISDNdcs values 5: 1124
IISDNdir values 5: 1036
IISDNds values 5: 1119
IISDNdsmsk values 5: 1122
IISDNepcmd values 5: 1035
IISDNfa values 5: 1131
IISDNl1mod values 5: 1036
IISDNl2err values 5: 1120
IISDNl3mod values 5: 1038
IISDNline_type values 5: 1095, 5: 1117
IISDNloc values 5: 1064
5: 861, 5: 868
IISDNlpds values 5: 1120
IISDNns values 5: 1014, 5: 1130
IISDNnump values
5: 856, 5: 858, 5: 864, 5: 872
IISDNnumt values 5: 858
5: 856, 5: 864, 5: 872
IISDNpres values 5: 873
5: 859
IISDNprog values 5: 869, 5: 1065
5: 869
IISDNrrsn values
5: 874
IISDNscr values
5: 859, 5: 873
IISDNsigtype values 5: 950
IISDNst values 5: 1038
IISDNvar values 5: 1039, 6: 1213, 6: 1236
IISDNvbf values 5: 1056
IISDNvbs values 5: 1093
Inbound call event sequence 5: 910
Incoming call
clearing or refusing 5: 1020
SETUP indication 5: 1129
Incoming call answering 2: 280
Incoming Call function
BfvCallHold 2: 289
BfvCallReject 2: 293
BfvCallRingDetect 2: 297
BfvCallWaitForHold 2: 345
BfvCallWaitForSetup 2: 352
Infopkt functions 3: 600–3: 625
closing current file 3: 601
closing prompt files 3: 622
giving pointer position 3: 605
handling user-defined infopkts 3: 619
opening prompt files 3: 624
opening stream disk files 3: 609
opening streams 3: 611
reading infopkt from stream 3: 607
replacing last infopkt in stream 3: 617
seeking to given offset 3: 603
writing to infopkt streams 3: 615
Infopkt header 6: 1399
Infopkt parameters
ASCII strip infopkt 6: 1404
beginning of page infopkt 6: 1417
document parameters infopkt 6: 1406
end-of-speech parameter infopkt 6: 1400

enhanced fax format page infopkt 6: 1408
fax header parameters infopkt 6: 1409
G3 strip infopkt 6: 1410
page parameters infopkt 6: 1412
prompt map infopkt 6: 1401
speech parameters infopkt 6: 1402
T.30 parameters infopkt 6: 1414
Information Elements (IEs)
appending to BSMI messages 5: 865
appending to SMI messages 5: 1082
common structure 5: 865
disabling error checking 5: 1044
locking codeset shifts 5: 865
user IE encoding 5: 1044
INFORMATION Q.931 message 5: 1060
Initialize channel 1: 65
Initializing call control 2: 284
Instant ISDN Software (IISDN) 5: 805
Interface, Ethernet parameters 6: 1286–6: 1290
Internet Aware Fax (IAF) 6: 1163
Internet protocol, see IP 6: 1245
Interrupt overrun status 1: 264
Interrupt processing, aborting 1: 215
IP
call control configuration 6: 1245–6: 1291
configuring 3rd party call control stack 6: 1291
customizing 3rd party call control stack 6: 1257
H.323 protocol parameters 6: 1258
SIP protocol parameters 6: 1257, 6: 1267
Ireland, dialing requirements 6: 1453
ISDN calls
billing change 5: 1093
billing rates 5: 1058
far end disconnect 5: 1105
far end ringing indication 5: 1087
receiving call establishment indication 5: 1102
using L4L3mCALL_REQUEST 5: 1018
ISDN services
call control functions 2: 277
ISDN supplemental messages summary, BSMI 5: 841
Israel, dialing requirements 6: 1454
Italy, dialing requirements 6: 1454
ITU-T switch type supported 5: 1051
ITU-T T.30 fax protocol specification 6: 1372

J

Japan, dialing requirements 6: 1454
JATE redial restriction
configuration 6: 1181, 6: 1182
JATE variant
switch type supported 5: 1051

K

Korea R2 parameter file 6: 1472

L

L3L4 common header 5: 832
L3L4 messages 5: 1087–5: 1137
L3L4mABCD_SIGNAL_DATA
LEC protocol 5: 970
L3L4mACK_DOWNLOAD
LEC protocol 5: 971
L3L4mACK_UPLOAD
LEC protocol 5: 972
L3L4mALERTING 5: 1087
LEC protocol 5: 973
R2 signaling 5: 913
L3L4mANI 5: 1088
L3L4mB_CHANNEL_STATUS 5: 1090
L3L4mBILLING STATUS 5: 1093
L3L4mBOARD_ID 5: 1094
L3L4mCALL 5: 1096
L3L4mCALL_PROC_SENT 5: 1097
L3L4mCALLER_ID_DETECTED
LEC protocol 5: 974
L3L4mCAS_CHAN_BLOCKED
R2 signaling 5: 914
L3L4mCAS_CHAN_UNBLOCKED
R2 signaling 5: 916
L3L4mCAS_SIGNALING_BIT_STATUS
LEC protocol 5: 977
R2 signaling 5: 917
L3L4mCAS_STATUS
R2 signaling 5: 919
L3L4mCLEAR_REQUEST 5: 1098
LEC protocol 5: 980
R2 signaling 5: 921
L3L4mCLEAR_WITH_RESTART_REQUEST 5: 1100

L3L4mCONFIGURATION_STATUS
LEC protocol 5: 982
L3L4mCONN_ACK_IND 5: 1103
LEC protocol 5: 983
R2 protocol 5: 923
R2 signaling 5: 923
L3L4mCONNECT 5: 1102
LEC protocol 5: 984
R2 signaling 5: 924
L3L4mD_CHANNEL_STATUS 5: 1104
L3L4mDISCONNECT 5: 1104
LEC protocol 5: 985
R2 signaling 5: 925
L3L4mERROR 5: 1106
message definitions 5: 1106
R2 signaling 5: 927
L3L4mINFO_REQUEST 5: 1111
L3L4mLINE_STATUS 5: 1116
L3L4mLOOP_ON
LEC protocol 5: 992
L3L4mLOOP_REVERSAL
LEC protocol 5: 994
L3L4mPRE_SEIZE
LEC protocol 5: 995
R2 signaling 5: 929
L3L4mPROGRESS 5: 1118
LEC protocol 5: 996
L3L4mPROTOCOL_STATUS 5: 1119
L3L4mRAW_QDATA 5: 1126
L3L4mRESTART 5: 1128
B-channel maintenance 5: 1142
L3L4mRING_STATUS
LEC protocol 5: 997
L3L4mRX_WINK
LEC protocol 5: 999
L3L4mSEIZE_COMP
LEC protocol 5: 1000
L3L4mSETUP_IND 5: 1129
LEC protocol 5: 1001
L3L4mSTATUS_IND 5: 1134
LEC protocol 5: 1002
L3L4mTX_HOOKFLASH_END
LEC protocol 5: 1003
L3L4mTXWINK_END
LEC protocol 5: 1004
L3L4mUNIVERSAL 5: 1135
L3L4mUSER_INFO 5: 1137
L4 reference value 5: 846
L4L3 common header 5: 832
L4L3 messages 5: 1006–5: 1085
L4L3mALERTING_REQUEST 5: 1006
L4L3mCALL_PROCEEDING 5: 1096
L4L3mCALL_PROCEEDING_REQUEST 5: 1008
L4L3mCALL_REQUEST 5: 1012
LEC protocol 5: 936
R2 signaling 5: 885
L4L3mCAS_CHAN_BLOCK
R2 signaling 5: 888
L4L3mCAS_CHAN_UNBLOCK
R2 signaling 5: 890
L4L3mCLEAR_REQUEST 5: 1020
LEC protocol 5: 939
R2 signaling 5: 892
L4L3mCOLLECT_DIGITS
LEC protocol 5: 941
R2 Signaling 5: 894
L4L3mCONNECT_REQUEST 5: 896
LEC protocol 5: 943
L4L3mDIAL
LEC protocol 5: 945
L4L3mDISABLE_B_CHANNEL 5: 1025
B-channel maintenance 5: 1141
L4L3mDISABLE_CAS
LEC protocol 5: 948
R2 signaling 5: 898
L4L3mDISABLE_D_CHANNEL 5: 1029
L4L3mDISABLE_PROTOCOL
D-channel maintenance 5: 1147
L4L3mENABLE_B_CHANNEL 5: 1030
L4L3mENABLE_CAS
LEC protocol 5: 950
R2 signaling 5: 900
L4L3mENABLE_PROTOCOL 5: 1033
command mode 5: 1035
D-channel maintenance 5: 1147
L4L3mEND_DIAL
LEC protocol 5: 953
L4L3mFACILITY_REQUEST 5: 1052
L4L3mFEATURE_REQUEST 5: 1055
L4L3mFORCE_CONNECTION_REQUEST
LEC protocol 5: 955
L4L3mINFO_REQUEST 5: 1059
L4L3mJATE_REDIAL 5: 1061
L4L3mPROGRESS_REQUEST 5: 1063
L4L3mREQ_ABCD_DATA
LEC protocol 5: 957

R2 signaling 5: 905
L4L3mREQ_CONFIGURATION
LEC protocol 5: 958
L4L3mREQ_L2_STATS 5: 1070
L4L3mREQ_LINE_STATUS 5: 1069
L4L3mREQ_PROTOCOL_STATUS 5: 1072
L4L3mRESTART 5: 1073
B_channel maintenance 5: 1141
L4L3mSET_CAS_SIGNALING_BITS
LEC protocol 5: 960
R2 signaling 5: 906
L4L3mSET_CONFIGURATION
LEC protocol 5: 960
L4L3mSET_HARDWARE 5: 1077
L4L3mSETUP_ACK_REQUEST 5: 1078
L4L3mTX_HOOKFLASH
LEC protocol 5: 962
L4L3mTX_WINK
LEC protocol 5: 964
L4L3mUNIVERSAL 5: 1081
L4L3mUSER_INFO 5: 1085
Layer 2 statistics 5: 1070, 5: 1113
LEC protocol
common fields 5: 968
LEC protocols 5: 934–5: 1004
analog configuration 6: 1199
E1 CAS configuration 6: 1219
T1 RBS configuration 6: 1196, 6: 1242
Line administration and initialization functions 1: 50–1: 81
Line characteristics, analog DID 6: 1195
Line data structure
preparing for TIFF-F files 4: 667
preparing for transmission 4: 662
Line firmware macros 1: 113–1: 115
Line information macro 1: 82, 1: 83, 3: 598
Line pointer information macro 1: 84
Line state
BTLINE structure 1: 47
LINE_ALERT_CTL macro 1: 84
LINE_AOC_INFO macro 2: 487
LINE_API_VER_LOADED() macro 1: 88
LINE_APP_ADDR macro 1: 90
LINE_CONFIG_STRUCT macro 1: 86
LINE_CPU_TYPE macro 1: 83
LINE_DCS macro 4: 778
LINE_DEST_ADDR macro 1: 90
LINE_DIS_DTC macro 4: 778
LINE_DRIVER_VER_LOADED macro 1: 89
LINE_ERR_INTR_DATA macro 1: 263
LINE_ERR_INTR_DATA_SIZE macro 1: 264
LINE_ERR_INTR_INTR_MSG macro 1: 263
LINE_ERROR_INTR macro 1: 263
LINE_FAX_RES macro 4: 779
LINE_FAX_T30_RCV_MCF_FSK macro 4: 779
LINE_FIRM_BITRATE macro 4: 779
LINE_FIRM_BOOT_ROM_AUTO_NUM macro 1: 111
LINE_FIRM_BOOT_ROM_BUILD macro 1: 111
LINE_FIRM_BOOT_ROM_COMMENT macro 1: 111
LINE_FIRM_BOOT_ROM_DATE macro 1: 111
LINE_FIRM_BOOT_ROM_MAJOR macro 1: 111
LINE_FIRM_BOOT_ROM_MIDDLE macro 1: 111
LINE_FIRM_BOOT_ROM_MINOR macro 1: 111
LINE_FIRM_CHECKSUM macro 1: 114
LINE_FIRM_CHK_OK macro 1: 115
LINE_FIRM_CTRL_PROC_AUTO_NUM macro 1: 112
LINE_FIRM_CTRL_PROC_BUILD macro 1: 112
LINE_FIRM_CTRL_PROC_COMMENT macro 1: 113
LINE_FIRM_CTRL_PROC_DATE macro 1: 112
LINE_FIRM_CTRL_PROC_MAJOR macro 1: 112
LINE_FIRM_CTRL_PROC_MIDDLE macro 1: 112
LINE_FIRM_CTRL_PROC_MINOR macro 1: 112
LINE_FIRM_DATE macro 1: 114
LINE_FIRM_DOWNLOADED macro 1: 115
LINE_FIRM_DSP_AUTO_NUM macro 1: 114
LINE_FIRM_DSP_BUILD macro 1: 113
LINE_FIRM_DSP_COMMENT macro 1: 114
LINE_FIRM_DSP_DATE macro 1: 114
LINE_FIRM_DSP_MAJOR macro 1: 113
LINE_FIRM_DSP_MIDDLE macro 1: 113
LINE_FIRM_DSP_MINOR macro 1: 113
LINE_FIRM_ID macro 1: 115
LINE_FIRM_MAJOR macro 1: 114
LINE_FIRM_MIDDLE macro 1: 114
LINE_FIRM_MINOR macro 1: 114
LINE_FIRM_NUM_DSPS macro 1: 113
LINE_FIRM_TYPE macro 1: 115
LINE_FONT_DOWNLOADED macro 4: 779
LINE_HAS_CAP macro 1: 83, 3: 598
LINE_INCOMING_CMD_FUNC macro 1: 91
LINE_INTR_OVERRUN macro 1: 264
LINE_PAGE_COMPLETE_ARG macro 4: 780

LINE_PAGE_COMPLETE_FUNC macro 4: 780
LINE_PAGE_CT macro 4: 780
LINE_PHONE_STRUCT macro 1: 86
LINE_PRIVATE_USER_DATA macro 1: 84
LINE_SET_INCOMING_CMD FUNC macro 1: 92
LINE_SET_PAGE_COMPLETE_FUNC macro 4: 780
LINE_SRC_ADDR macro 1: 90
LINE_STATE macro 1: 82
LINE_TYPE macro 1: 83
LINE_UNIT_NUM macro 1: 83
LINE_VAD_BYTES_PROCESSED macro 3: 598
LINE_VAD_STATE macro 3: 598
Link Layer 2 information 5: 1122
Loaded driver version macro 1: 89
Local Access and Transport Area (LATA) 5: 931
Local Exchange Carriers (LEC) 5: 931
Local ID string parameters 6: 1163, 6: 1164
Locking codeset shifts for IEs 5: 865
Logical Link ID, defined 5: 849
Loop current
  turning off detection 2: 416
  turning on detection 2: 418
Loop start, analog
  port configuration 6: 1197
Low-level call control function
  BfvCallCtrlClose 2: 283
  BfvCallCtrlInit 2: 284
  BfvCallDisconnect 2: 287
  BfvCallHold 2: 289
  BfvCallReconfigureHostModule 2: 291
  BfvCallReject 2: 293
  BfvCallRetrieve 2: 295
  BfvCallRingDetect 2: 297
  BfvCallSendAlerting 2: 301
  BfvCallSetup 2: 303
  BfvCallStatus 2: 326
  BfvCallTransferComplete 2: 329
  BfvCallWaitForAccept 2: 331
  BfvCallWaitForAlerting 2: 334
  BfvCallWaitForComplete 2: 337
  BfvCallWaitForHold 2: 345
  BfvCallWaitForRelease 2: 347
  BfvCallWaitForRetrieve 2: 350
  BfvCallWaitForSetup 2: 352
  BfvCallWaitTransferComplete 2: 359
Low-level packet sending function 1: 212
LP_BOARD_SLOT macro 1: 84
LP_CPU_NUM macro 1: 84

M

Macros
  administration and initialization 1: 82, 2: 487
  firmware 1: 111
  Millennial low-level 1: 90
  TIFF 4: 801
  VAD 3: 598
Malaysia, dialing requirements 6: 1455
Management messages summary, BSMI 5: 833
MD-110 5: 1051
Memory allocation 1: 218
Mexico R2 parameter file 6: 1475
MF tones mapping 3: 516
MILL_API_BUILD_NUM macro 1: 87
MILL_BUILD_NUM macro 1: 88
MILL_DRIVER_VERSION macro 1: 88
MILL_V1 macro 1: 88
MILL_V3 macro 1: 89
MILL_V4 macro 1: 89
MILL_VER_NUM macro 1: 88
Millennial environment macro 1: 86
Millennium commands function macro 1: 91
Miscellaneous message summary, BSMI 5: 841
MOD_BOARD_SLOT macro 1: 83
MOD_CPU_NUM macro 1: 83
Modem calls 5: 1129
Module configuration 6: 1178
Module information 1: 71
Module number information macro 1: 83
Monitoring call signaling state 2: 319
Multibyte storage formats macros 1: 85

N

Name, caller 6: 1352, 6: 1354
National ISDN-1 variant, switch type supported 5: 1051
National ISDN-2 variant
  switch type supported 5: 1051
NET-5 variant
  standard 5: 1039, 6: 1213, 6: 1236
  switch type supported 5: 1051
Netherlands, dialing requirements 6: 1455
Network side

emulation 5: 1042
signaling 5: 1036
New Zealand, dialing requirements 6: 1455
Non-Facilities Associated Signaling (NFAS)
5: 1009, 5: 1015, 5: 1026, 5: 1074
configuring a D-channel 5: 1042
D-channel backup 5: 1125
Noninfopkt raw files
initiating 4: 647
opening/reading/transferring specified files 4: 738
preparing line data structure 4: 662
receiving G3 pages 4: 721
receiving noninfopkt fax page to a file 4: 725
sending end-of-page 4: 698
sending strip parameters 4: 759
separating data strips 4: 759
setting page parameters 4: 704
transferring ASCII/G3 data 4: 736
Nortel
Custom variant 5: 1051
DMS-100 switch 5: 1051
DMS-250 switch 5: 1038, 5: 1051
services 5: 1014, 5: 1130
Norway, dialing requirements 6: 1456
NTT switch type 5: 1051
Number of DSPs macro 1: 113
Numbering conventions
LEC protocols 5: 933
R2 signaling 5: 878

O

On-hook state 2: 392
Originating number (ANI) 5: 1088
Outbound call event sequence 5: 911
Outgoing call
initiating using SETUP 5: 1012
progress indication 5: 1118
setting signaling state 2: 323
setting up 2: 303
Outgoing Call function
BfvCallSetup 2: 303
BfvCallWaitForComplete 2: 337

P

Packets
receiving and processing 1: 221
Page completion information macros 4: 780
Page number, resetting 4: 689
Page parameters
infopkt 6: 1412
maximum amount to store 6: 1164
maximum page width 6: 1164
minimum number of lines 6: 1165
recording in infopkt stream 4: 720
setting 4: 704
specifying results from mismatching 6: 1169
Page resolution
fax reception 6: 1166
in TIFF-F files 4: 799
Page Result structure parameters 6: 1357
PAGE_RES parameters 6: 1357
Parameters
analog DID call control 6: 1194
analog DID line characteristics 6: 1195
analog loop start call control 6: 1197
BRI call control 6: 1201
call control configuration 6: 1170–6: 1291
country-specific 6: 1441
customizing IP call control stack 6: 1257
E1 CAS call control 6: 1217
E1 CAS R2 call control 6: 1221
E1 ISDN call control 6: 1209
E1 QSIG call control 6: 1223
Ethernet interface 6: 1286–6: 1290
IP call control configuration 6: 1245–6: 1291
port configuration 6: 1190–6: 1196, ??–6: 1243
predefined H.323 IP protocol 6: 1258
predefined SIP IP protocol 6: 1257, 6: 1267
T.38 fax configuration 6: 1248
T1 ISDN call control 6: 1232
T1 RBS call control 6: 1240
Phase C codes 6: 1383
Placing
line on-hook 2: 392
outgoing call 2: 303
Polling variations 4: 706
Port configuration 6: 1190–6: 1196, ??–6: 1243
pragma pack, see Chapter 1
Predefined H.323 parameters 6: 1258
Predefined SIP parameters 6: 1257, 6: 1267
Primary Rate (PRI)
variants supported 5: 1050
Procedure interrupt macros 4: 780

Progress Indication common structure 5: 867
PROGRESS Q.931 5: 1118
L3L4mCONNECT 5: 1102
L4L3mPROGRESS_REQUEST 5: 1063
Prompt files
closing 3: 622
opening 3: 624
playing phrases 3: 531
Prompt map infopkt 6: 1401
Protocol variants 6: 1213, 6: 1236
Pulse dialing mode 6: 1167

Q

Q.931 Layer 3 protocol timers 5: 1041
Q.931 messages
ALERTING 5: 1087, 5: 1102
B-channel maintenance 5: 1140
CALL PROCEEDING 5: 1011, 5: 1133
CONNECT 5: 1102
DISCONNECT 5: 1104
FACILITY 5: 1057
INFORMATION 5: 1060
PROGRESS 5: 1067, 5: 1102, 5: 1118
raw data 5: 1126
REGISTER 5: 1082
RELEASE 5: 1099
RELEASE COMPLETE 5: 1082, 5: 1099
RESTART 5: 1075, 5: 1101, 5: 1125, 5: 1128
RESTART ACK 5: 1101
SERVICE 5: 1032, 5: 1092
5: 1032
SERVICE ACK 5: 1027, 5: 1032, 5: 1092
SETUP 5: 1012, 5: 1133
SETUP ACKNOWLEDGE 5: 1080
STATUS 5: 1134
Q.931 protocol errors 5: 1134
Q.933 DLCI Negotiation common structure 5: 870
Query, transfer capability 2: 404

R

R2 parameter file
Argentina 6: 1459
Brazil 6: 1462
China 6: 1469
configuring for port use 6: 1222
Korea 6: 1472
Mexico 6: 1475
R2 protocol
inbound call event sequence 5: 910
message summary 5: 877
module to host events 5: 908, 5: 966
numbering conventions 5: 878
outbound call event sequence 5: 911
Rate adaption value 5: 1036
Raw Q.931 message 5: 1043, 5: 1126
Receive phase B codes 6: 1380
Receive phase D codes 6: 1382
Receiving and processing packets 1: 221
Reconfiguring host module 2: 291
Redial restriction, JATE 6: 1181, 6: 1182
Redirecting Number common structure 5: 871
Rejecting incoming call 2: 293
Release Link Trunk (RLT) signaling 2: 329, 2: 399, 2: 403, 2: 407, 2: 447, 2: 479, 5: 1053, 6: 1239
Releasing a call 2: 347
RES structure parameters 6: 1348
RESTART ACK Q.931 message 5: 1101
Result structures 6: 1344–6: 1370, 6: 1485–??
CALL_RES parameters 6: 1352
decoded DCS, DIS, DTC info structures 6: 1367
FAX_RES parameters 6: 1362
file location 6: 1344
Page result structure parameters 6: 1357
PAGE_RES parameters 6: 1357
RES structure parameters 6: 1348
Retrieving
calling party 2: 402
channel call state 2: 326
Return of error message strings 1: 244
Reverse
long byte order 1: 85
short byte order 1: 85
Ring detection 2: 297

S

SABME message 5: 1037, 5: 1122
Send Alerting message 2: 301
Service Access Point Identifier (SAPI) 5: 849
SERVICE Q.931 message 5: 1030
Session Millennial address macro 1: 90

Sessions, closing 1: 81
Setting call signaling state 2: 323
Setting ring detection 2: 297
Setting state to on-hook 2: 392
Setting variables 1: 224
Shutting down call control 2: 283, 2: 291
Siemens switch
  switch type 5: 1039
  variants supported 5: 1051
Signal detection and generation
  function details 3: 492–3: 512
  function summary 3: 491
Signaling state
  monitoring inbound call 2: 319
  setting for outbound call 2: 323
Singapore, dialing requirements 6: 1456
SIP IP call control parameters 6: 1257, 6: 1267
Software Defined Network (SDN) 5: 1130
Source Millennial address macro 1: 90
Spain, dialing requirements 6: 1456
Special information tones 6: 1434
  SITINTC 6: 1434
  SITNOCIR 6: 1434
  SITREORD 6: 1435
  SITVACODE 6: 1435
Speech
  initial gain value for playback 6: 1167
  modifying properties 3: 535
  parameters infopkt 6: 1402
  playing from infopkt stream 3: 541
  playing from prompt files 3: 531
  playing raw speech data 3: 544, 3: 551
  playing speech from a wave file 3: 557
  recording in infopkt format 3: 561
  recording raw data 3: 580
  recording raw speech data 3: 570
  recording speech 3: 589
  voice encoding settings 3: 559, 3: 566, 3: 568, 3: 576, 3: 586, 3: 594
Speed control 3: 539
Stack to application events 5: 908, 5: 966
Starting outgoing call 2: 303
Status function
  BfvCallStatus 2: 326
Storage formats macros 1: 85
Strip parameters
  sending 4: 759
Structure packing, see Chapter 1
Switch to voice mode 4: 682
Switch type 5: 1038
  variant matrix 5: 1050
switch type supported 5: 1051
Switzerland, dialing requirements 6: 1457

T

T.30
  enabling holdup 4: 763
  infopkt 6: 1414
  interrupt macro 4: 780
  ITU-T fax specification 6: 1372
  negotiation holdup 4: 765
  setting bit rate/scan time 4: 769
T.38 Fax transport parameters 6: 1248
T1 ISDN
  port configuration 6: 1232
T1 Robbed Bit
  port configuration 6: 1240
TBR 4 standard
  boards approved 6: 1452
teleph.cfg file 2: 272, 6: 1150
Telephony functions 1: 163–1: 168
Terminal Endpoint Identifier (TEI) 5: 849
Terminating call 2: 287, 2: 347
Third party IP call control stack
  configuring custom parameters 6: 1257
  configuring for module use 6: 1291
  predefined H.323 parameters 6: 1258–6: 1262
  predefined SIP parameters 6: 1267–6: 1279
  rereading configuration file 2: 291
TIFF_FP macro 4: 801
TIFF-F files
  file pointer macro 4: 801
  initiating when polling is required 4: 670
  interpreting IFD entries 4: 792
  opening 4: 785
  preparing line data structure 4: 667
  reading IFD 4: 787
  reading image data 4: 790
  sending 4: 701
  transferring G3 data 4: 713
  transmitting pages from 4: 746
  writing IFD of current page 4: 794
  writing image data 4: 797
  writing page resolution data 4: 799

Timeout values 6: 1164
Timers
  Layer 2 timers 5: 1038, 5: 1039, 6: 1213, 6: 1236
Tone detection
  function details 3: 513–??
  function summary 3: 491
Tones
  available for next call 3: 528
  backing up more than one tone 3: 528
  discarding 3: 518
  DTMF dialing mode 6: 1167
  generating/playing groups 3: 495
  generating/playing patterns 3: 495
  next in buffer 3: 521
  playing for specified time 3: 523
  playing single frequency 3: 525
  removing tones from toner buffer 3: 519
Tool, Call Tracer 1: 231
TR series boards, approvals 6: 1439
Trace options, configuring call control 6: 1176
Tracer utility, Call 1: 231
Transfer
  canceling call 2: 402
  channel capability 2: 404
  completing call connection 2: 406
Transfer call completion 2: 329
transfer_variant values 6: 1200, 6: 1208, 6: 1215, 6: 1220, 6: 1231, 6: 1239, 6: 1243
Transferring call 2: 303, 2: 396
Transition
  out of hold state 2: 295, 2: 350
  to hold state 2: 289, 2: 345
Transmission priority 5: 1037
Transmit codes
  phase A 6: 1373
  phase B 6: 1374
  phase D 6: 1376
TS014 variant 5: 1051
Turkey, dialing requirements 6: 1457

U

Undecoded Q.931 packets 5: 1126
United Kingdom, dialing requirements 6: 1457
United States, dialing requirements 6: 1458
Unknown switch type 5: 1051
User Info common structure 5: 875
User-defined configuration file 1: 63, 1: 117, 2: 279, 6: 1152
  file location 6: 1153
  keywords 6: 1153
  parameters 6: 1153
User-side ISDN signaling 5: 1036
User-user IE 5: 875
USES_FAT_FILESYSTEM macro 1: 85
Utility Function messages summary, BSMI 5: 842
Utility, Call Tracer 1: 231

V

V.110 calls 5: 1015
Variabill feature 5: 1093
Variables, setting 1: 224
Variants
  call transfer 6: 1200, 6: 1208, 6: 1215, 6: 1220, 6: 1231, 6: 1239, 6: 1243
  protocol 6: 1213, 6: 1236
Version information macros 1: 86, 1: 87, 1: 88
VN3 France variant 5: 1039, 6: 1213
VN3 variant, supported switch types 5: 1051
Voice Activity Detection (VAD) 3: 574, 3: 584, 3: 592
Voice encoding settings 3: 559, 3: 566, 3: 568, 3: 576, 3: 586, 3: 594
Voice mode, switching 4: 682
Voice play and record
  function details 3: 531–3: 597
  function summary 3: 530
Volume control 3: 539

W

Wait time 2: 431
Waiting
  for answer to outbound call 2: 337
  to clear call 2: 347
  to complete call holding process 2: 345
  to detect incoming call 2: 352, 2: 408
  to finish answering call 2: 331
  to finish establishing or dialing outbound 2: 334
Wink support, BSMI 5: 965