
MFi Accessory Firmware Specification

Release R43



2011-04-04



Apple Inc.
© 2011 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

App Store is a service mark of Apple Inc.

iTunes Store is a registered service mark of Apple Inc.

Apple, the Apple logo, FireWire, iPhone, iPod, iPod classic, iPod nano, iPod shuffle, iPod touch, iTunes, Mac, Mac OS, Macintosh, Numbers, Pages, and Spotlight are trademarks of Apple Inc., registered in the United States and other countries.

iPad and QuickStart are trademarks of Apple Inc.

iOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

Times is a registered trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction Introduction 31

Organization of This Document 37
General Specification Terms 38
Special Terminology 38
See Also 38

Chapter 1 Protocol Features and Availability 41

General Apple Device Features 44
Audio and Video Output Preferences 47
 Video Output Settings 48
Character Encoding 49
Accessory Control of iOS Devices 49
Accessory Launching of iOS Applications 49
Accessory Communication With iOS Applications 50
 Setting Up a Communication Session 51
 Data Flow to and from the iOS Device 52
 Matching Accessories With Applications 52
 Communication Protocol Design Hints 52
 Communication iAP Commands 53
 Controlling Applications Using the Simple Remote Lingo 53
Using an Apple Device as a USB Host 54
 Entering USB Host Mode on Startup 54
 Upgrading from UART Transport to USB Host Mode 55
 Packet Formats for iAP Commands 56
 Data Transfers With an Apple Device 56
 Optimizing Data Transfers 58
iPod Out Mode 59
 Setting iPod Out Video Preferences 61
 Supported Simple Remote and Display Remote Commands 62
Improving User Interface Accessibility 63

Chapter 2 The Protocol Core and the General Lingo 65

Reserved Commands and Data 65
Accessory Signaling and Initialization 66
 Packet Signaling and Initialization Using the UART Serial Port Link 66
 iAP Signaling and Initialization Using the USB or BT Port Link 69
Authentication 71
 Levels of Accessory Authentication 71
 Authentication Requirements 72

Apple Device Authentication of the Accessory	73
Accessory Authentication of the Apple Device	75
Command Packet Formats	76
Small Packet Format	77
Large Packet Format	77
Packet Details	78
Lingo 0x00: General Lingo	78
General Lingo Command Summary	79
History of the General lingo protocol	82
Command 0x00: RequestIdentify	83
Command 0x02: ACK	83
Command 0x03: RequestRemoteUIMode	88
Command 0x04: ReturnRemoteUIMode	88
Command 0x05: EnterRemoteUIMode	89
Command 0x06: ExitRemoteUIMode	90
Command 0x07: RequestiPodName	90
Command 0x08: ReturniPodName	91
Command 0x09: RequestiPodSoftwareVersion	91
Command 0x0A: ReturniPodSoftwareVersion	92
Command 0x0B: RequestiPodSerialNum	93
Command 0x0C: ReturniPodSerialNum	93
Command 0x0F: RequestLingoProtocolVersion	94
Command 0x10: ReturnLingoProtocolVersion	94
Command 0x11: RequestTransportMaxPayloadSize	95
Command 0x12: ReturnTransportMaxPayloadSize	95
Command 0x13: IdentifyDeviceLingoes	96
Command 0x14: GetDevAuthenticationInfo	100
Command 0x15: RetDevAuthenticationInfo	100
Command 0x16: AckDevAuthenticationInfo	102
Command 0x17: GetDevAuthenticationSignature	103
Command 0x18: RetDevAuthenticationSignature	103
Command 0x19: AckDevAuthenticationStatus	104
Command 0x1A: GetiPodAuthenticationInfo	105
Command 0x1B: RetiPodAuthenticationInfo	105
Command 0x1C: AckiPodAuthenticationInfo	106
Command 0x1D: GetiPodAuthenticationSignature	107
Command 0x1E: RetiPodAuthenticationSignature	107
Command 0x1F: AckiPodAuthenticationStatus	108
Command 0x23: NotifyiPodStateChange	109
Command 0x24: GetiPodOptions	110
Command 0x25: RetiPodOptions	110
Command 0x27: GetAccessoryInfo	111
Command 0x28: RetAccessoryInfo	114
Command 0x29: GetiPodPreferences	120
Command 0x2A: RetiPodPreferences	125
Command 0x2B: SetiPodPreferences	125

Command 0x35: GetUIMode	127
Command 0x36: RetUIMode	127
Command 0x37: SetUIMode	128
Command 0x38: StartIDPS	128
Command 0x39: SetFIDTokenValues	130
Command 0x3A: RetFIDTokenValueACKs	139
Command 0x3B: EndIDPS	143
Command 0x3C: IDPSStatus	144
Command 0x3F: OpenDataSessionForProtocol	146
Command 0x40: CloseDataSession	147
Command 0x41: DevACK	147
Command 0x42: DevDataTransfer	148
Command 0x43: iPodDataTransfer	150
Command 0x46: SetAccStatusNotification	152
Command 0x47: RetAccStatusNotification	153
Command 0x48: AccessoryStatusNotification	154
Command 0x49: SetEventNotification	157
Command 0x4A: iPodNotification	159
Command 0x4B: GetiPodOptionsForLingo	165
Command 0x4C: RetiPodOptionsForLingo	166
Command 0x4D: GetEventNotification	176
Command 0x4E: RetEventNotification	176
Command 0x4F: GetSupportedEventNotification	177
Command 0x50: CancelCommand	178
Command 0x51: RetSupportedEventNotification	179
Command 0x54: SetAvailableCurrent	180
Command 0x64: RequestApplicationLaunch	181
Command 0x65: GetNowPlayingFocusApp	181
Command 0x66: RetNowPlayingFocusApp	182

Chapter 3**Accessory Lingoes 183**

Command Timings	184
Lingo 0x02: Simple Remote Lingo	185
History and Applicability	185
Playback Engine Playlists	188
Using Contextual Buttons	188
Using Dedicated Media Buttons	190
Accessory Control of the iPod 5G nano Camera	190
USB Human Interface Device Reports	197
User Interface Accessibility Commands	198
Command 0x00: ContextButtonStatus	198
Command 0x01: ACK	200
Command 0x03: VideoButtonStatus	201
Command 0x04: AudioButtonStatus	203
Command 0x0B: iPodOutButtonStatus	205

Command 0x0C: RotationInputStatus	206
Command 0x0D: RadioButtonStatus	210
Command 0x0E: CameraButtonStatus	211
Command 0x0F: RegisterDescriptor	212
Command 0x10: SendHIDReportToiPod	215
Command 0x11: SendHIDReportToAcc	215
Command 0x12: UnregisterDescriptor	216
Command 0x13: AccessibilityEvent	217
Command 0x14: GetAccessibilityParameter	219
Command 0x15: RetAccessibilityParameter	220
Command 0x16: SetAccessibilityParameter	221
Command 0x17: GetCurrentItemProperty	221
Command 0x18: RetCurrentItemProperty	222
Command 0x19: SetContext	224
Command 0x1A: AccParameterChanged	225
Command 0x81: DevACK	226
Lingo 0x03: Display Remote Lingo	226
Command History of the Display Remote lingo	229
Transferring Album Art	229
Command 0x00: ACK	230
Command 0x01: GetCurrentEQProfileIndex	231
Command 0x02: RetCurrentEQProfileIndex	231
Command 0x03: SetCurrentEQProfileIndex	232
Command 0x04: GetNumEQProfiles	233
Command 0x05: RetNumEQProfiles	233
Command 0x06: GetIndexedEQProfileName	234
Command 0x07: RetIndexedEQProfileName	235
Command 0x08: SetRemoteEventNotification	235
Command 0x09: RemoteEventNotification	237
Command 0x0A: GetRemoteEventStatus	244
Command 0x0B: RetRemoteEventStatus	245
Command 0x0C: GetiPodStateInfo	246
Command 0x0D: RetiPodStateInfo	247
Command 0x0E: SetiPodStateInfo	249
Command 0x0F: GetPlayStatus	254
Command 0x10: RetPlayStatus	254
Command 0x11: SetCurrentPlayingTrack	255
Command 0x12: GetIndexedPlayingTrackInfo	256
Command 0x13: RetIndexedPlayingTrackInfo	257
Command 0x14: GetNumPlayingTracks	259
Command 0x15: RetNumPlayingTracks	260
Command 0x16: GetArtworkFormats	260
Command 0x17: RetArtworkFormats	261
Command 0x18: GetTrackArtworkData	262
Command 0x19: RetTrackArtworkData	263
Command 0x1A: GetPowerBatteryState	264

Command 0x1B: RetPowerBatteryState	265
Command 0x1C: GetSoundCheckState	266
Command 0x1D: RetSoundCheckState	266
Command 0x1E: SetSoundCheckState	267
Command 0x1F: GetTrackArtworkTimes	267
Command 0x20: RetTrackArtworkTimes	268
Command 0x21: CreateGeniusPlaylist	269
Command 0x22: IsGeniusAvailableForTrack	270
Lingo 0x04: Extended Interface Lingo	271
Lingo 0x05: Accessory Power Lingo	271
Command History of the Accessory Power lingo	272
Command 0x02: BeginHighPower	272
Command 0x03: EndHighPower	273
Lingo 0x06: USB Host Mode Lingo	273
Command History of the USB Host Lingoes	274
Command 0x00: DevACK	274
Command 0x04: NotifyUSBMode	275
Command 0x80: iPodACK	276
Command 0x81: GetiPodUSBMode	277
Command 0x82: RetiPodUSBMode	277
Command 0x83: SetiPodUSBMode	278
Lingo 0x07: RF Tuner Lingo	278
RF Tuner Accessory Design	279
RF Tuner Power	280
RF Tuner Lingo Commands	280
Command History of the RF Tuner Lingo	282
Command 0x00: ACK	282
Command 0x01: GetTunerCaps	284
Command 0x02: RetTunerCaps	285
Command 0x03: GetTunerCtrl	287
Command 0x04: RetTunerCtrl	287
Command 0x05: SetTunerCtrl	288
Command 0x06: GetTunerBand	289
Command 0x07: RetTunerBand	290
Command 0x08: SetTunerBand	291
Command 0x09: GetTunerFreq	291
Command 0x0A: RetTunerFreq	292
Command 0x0B: SetTunerFreq	293
Command 0x0C: GetTunerMode	294
Command 0x0D: RetTunerMode	294
Command 0x0E: SetTunerMode	295
Command 0x0F: GetTunerSeekRssi	296
Command 0x10: RetTunerSeekRssi	296
Command 0x11: SetTunerSeekRssi	297
Command 0x12: TunerSeekStart	297
Command 0x13: TunerSeekDone	300

Command 0x14: GetTunerStatus	300
Command 0x15: RetTunerStatus	301
Command 0x16: GetStatusNotifyMask	302
Command 0x17: RetStatusNotifyMask	302
Command 0x18: SetStatusNotifyMask	303
Command 0x19: StatusChangeNotify	304
Command 0x1A: GetRdsReadyStatus	306
Command 0x1B: RetRdsReadyStatus	306
Command 0x1C: GetRdsData	307
Command 0x1D: RetRdsData	308
Command 0x1E: GetRdsNotifyMask	310
Command 0x1F: RetRdsNotifyMask	311
Command 0x20: SetRdsNotifyMask	312
Command 0x21: RdsReadyNotify	313
Command 0x25: GetHDProgramServiceCount	314
Command 0x26: RetHDProgramServiceCount	314
Command 0x27: GetHDProgramService	315
Command 0x28: RetHDProgramService	315
Command 0x29: SetHDProgramService	316
Command 0x2A: GetHDDDataReadyStatus	317
Command 0x2B: RetHDDDataReadyStatus	317
Command 0x2C: GetHDDData	318
Command 0x2D: RetHDDData	319
Command 0x2E: GetHDDDataNotifyMask	321
Command 0x2F: RetHDDDataNotifyMask	322
Command 0x30: SetHDDDataNotifyMask	323
Command 0x31: HDDDataReadyNotify	324
Sample HD Command Sequences	326
Lingo 0x08: Accessory Equalizer Lingo	330
Equalizer Setting Requirements	330
Accessory Equalizer Lingo Commands	330
Command 0x00: ACK	331
Command 0x01: GetCurrentEQIndex	332
Command 0x02: RetCurrentEQIndex	332
Command 0x03: SetCurrentEQIndex	333
Command 0x04: GetEQSettingCount	333
Command 0x05: RetEQSettingCount	334
Command 0x06: GetEQIndexName	335
Command 0x07: RetEQIndexName	335
Lingo 0x09: Sports Lingo	336
Sports Lingo commands	336
Command History of the Sports Lingo	337
Command 0x00: DeviceACK	337
Command 0x01: GetDeviceVersion	338
Command 0x02: RetDeviceVersion	339
Command 0x03: GetDeviceCaps	339

Command 0x04: RetDeviceCaps	340
Command 0x80: iPodACK	341
Command 0x83: GetiPodCaps	342
Command 0x84: RetiPodCaps	342
Command 0x85: GetUserIndex	343
Command 0x86: RetUserIndex	344
Command 0x88: GetUserData	344
Command 0x89: RetUserData	345
Command 0x8A: SetUserData	347
Lingo 0x0A: Digital Audio Lingo	348
Accessory Authentication	349
Digital Audio Lingo Commands	349
Command History of the Digital Audio Lingo	350
USB Device Mode Audio	350
Command 0x00: AccAck	356
Command 0x01: iPodAck	357
Command 0x02: GetAccSampleRateCaps	358
Command 0x03: RetAccSampleRateCaps	358
Command 0x04: TrackNewAudioAttributes	360
Command 0x05: SetVideoDelay	361
Lingo 0x0C: Storage Lingo	362
Command History of the Storage Lingo	362
Command Summary	362
Command 0x00: iPodACK	363
Command 0x01: GetiPodCaps	365
Command 0x02: RetiPodCaps	365
Command 0x04: RetiPodFileHandle	367
Command 0x07: WriteiPodFileData	367
Command 0x08: CloseiPodFile	368
Command 0x10: GetiPodFreeSpace	369
Command 0x11: RetiPodFreeSpace	369
Command 0x12: OpeniPodFeatureFile	370
Command 0x80: DeviceACK	372
Command 0x81: GetDeviceCaps	373
Command 0x82: RetDeviceCaps	373
Lingo 0x0D: iPod Out Lingo	374
Command History of the iPod Out Lingo	374
iPod Out Lingo Commands	374
iPod Out Lingo Command Details	375
Lingo 0x0E: Location Lingo	380
Location Data Requirements	381
Power from the Apple Device for Accessories Using the Location Lingo	381
Command Summary	381
A Typical Location Data Session	383
Command 0x00: DevACK	386
Command 0x01: GetDevCaps	387

Command 0x02: RetDevCaps	388
Command 0x03: GetDevControl	390
Command 0x04: RetDevControl	391
Command 0x05: SetDevControl	392
Command 0x06: GetDevData	393
Command 0x07: RetDevData	395
Command 0x08: SetDevData	397
Command 0x09: AsyncDevData	400
Command 0x80: iPodACK	401
Sample Identification Sequences	402

Chapter 4 **The Extended Interface Protocol 409**

Major Apple Device Operating Modes	409
Extended Interface Mode	409
Sleep State	410
Packet Formats	411
Small Packet Format	411
Large Packet Format	411
Packet Details	412
Accessory Identification and Authentication	412
Command Transports	415
Entering Extended Interface Mode	415
Using the Extended Interface Protocol	416
The Playback Engine	417
The Database Engine	418
Transferring Album Art	422
Using the Extended Interface Protocol: An Example	423
Video Browsing	426
Sample Extended Interface Session	432
Command Packets	435
Command Code Summary	435
Command 0x0001: ACK	440
Command 0x0002: GetCurrentPlayingTrackChapterInfo	440
Command 0x0003: ReturnCurrentPlayingTrackChapterInfo	441
Command 0x0004: SetCurrentPlayingTrackChapter	442
Command 0x0005: GetCurrentPlayingTrackChapterPlayStatus	443
Command 0x0006: ReturnCurrentPlayingTrackChapterPlayStatus	444
Command 0x0007: GetCurrentPlayingTrackChapterName	444
Command 0x0008: ReturnCurrentPlayingTrackChapterName	445
Command 0x0009: GetAudiobookSpeed	446
Command 0x000A: ReturnAudiobookSpeed	446
Command 0x000B: SetAudiobookSpeed	447
Command 0x000C: GetIndexedPlayingTrackInfo	448
Command 0x000D: ReturnIndexedPlayingTrackInfo	450
Command 0x000E: GetArtworkFormats	453

Command 0x000F: RetArtworkFormats	454
Command 0x0010: GetTrackArtworkData	455
Command 0x0011: RetTrackArtworkData	456
Command 0x0016: ResetDBSelection	457
Command 0x0017: SelectDBRecord	458
Command 0x0018: GetNumberCategorizedDBRecords	460
Command 0x0019: ReturnNumberCategorizedDBRecords	461
Command 0x001A: RetrieveCategorizedDatabaseRecords	462
Command 0x001B: ReturnCategorizedDatabaseRecord	463
Command 0x001C: GetPlayStatus	464
Command 0x001D: ReturnPlayStatus	464
Command 0x001E: GetCurrentPlayingTrackIndex	465
Command 0x001F: ReturnCurrentPlayingTrackIndex	466
Command 0x0020: GetIndexedPlayingTrackTitle	467
Command 0x0021: ReturnIndexedPlayingTrackTitle	467
Command 0x0022: GetIndexedPlayingTrackArtistName	468
Command 0x0023: ReturnIndexedPlayingTrackArtistName	469
Command 0x0024: GetIndexedPlayingTrackAlbumName	470
Command 0x0025: ReturnIndexedPlayingTrackAlbumName	470
Command 0x0026: SetPlayStatusChangeNotification	471
Command 0x0027: PlayStatusChangeNotification	473
Command 0x0028: PlayCurrentSelection	475
Command 0x0029: PlayControl	476
Command: 0x002A: GetTrackArtworkTimes	478
Command: 0x002B: RetTrackArtworkTimes	479
Command 0x002C: GetShuffle	480
Command 0x002D: ReturnShuffle	480
Command 0x002E: SetShuffle	481
Command 0x002F: GetRepeat	483
Command 0x0030: ReturnRepeat	483
Command 0x0031: SetRepeat	484
Command 0x0032: SetDisplayImage	485
Command 0x0033: GetMonoDisplayImageLimits	491
Command 0x0034: ReturnMonoDisplayImageLimits	492
Command 0x0035: GetNumPlayingTracks	492
Command 0x0036: ReturnNumPlayingTracks	493
Command 0x0037: SetCurrentPlayingTrack	494
Command 0x0039: GetColorDisplayImageLimits	494
Command 0x003A: ReturnColorDisplayImageLimits	495
Command 0x003B: ResetDBSelectionHierarchy	496
Command 0x003C: GetDBiTunesInfo	497
Command 0x003D: RetDBiTunesInfo	498
Command 0x003E: GetUIDTrackInfo	499
Command 0x003F: RetUIDTrackInfo	501
Command 0x0040: GetDBTrackInfo	504
Command 0x0041: RetDBTrackInfo	505

Command 0x0042: GetPBTrackInfo	506
Command 0x0043: RetPBTrackInfo	507
Command 0x0044: CreateGeniusPlaylist	508
Command 0x0045: RefreshGeniusPlaylist	509
Command 0x0047: IsGeniusAvailableForTrack	510
Command 0x0048: GetPlaylistInfo	511
Command 0x0049: RetPlaylistInfo	512
Known Issues	513
Protocol History	513

Appendix A Accessory Identification 519

Using IDPS	519
IDPS Commands	520
Sample IDPS Command Sequences	520
Apple Device Event Notifications	523

Appendix B Transaction IDs 525

Using Transaction IDs	525
Generating and Returning Transaction IDs	525
Enabling and Disabling Transaction ID Support	525
Apple Device Acknowledgment of Transaction ID Commands	526
Applicability of Transaction IDs	526
Examples of Transaction IDs	527

Appendix C Multisection Data Transfers 531

Multisection Transfer Process	531
Multisection iPodACK Command	532
Multisection DevACK Command	532

Appendix D iTunes Tagging 535

The iTunes Tagging Experience	535
Tagging Feature Components	535
Broadcast Reception Accessory Requirements	537
Capturing and Storing Tag Data	537
Resolving Tag Ambiguity	538
Handling Unknown Metadata Types	539
Tag Data Writing Process	539
Data Transfer to the Apple Device	541
Accessory User Interface	544
Tag Button Text	546
Sample Command Sequence	546
FM Radio Tagging	550

The RT+ ODA	551
The iTunes Tagging ODA	552
Use of the RDS Group Type 3A	553
Tagging Application Group	554
Normal Mode and Test Mode	554
FM Tagging Event Control	555
FM Tag Decryption	555
HD Radio Tagging	560
Satellite Radio Tagging	561
Other Broadcast Tagging	562
Sample Tag Files	562
FM Radio Sample	562
HD Radio Samples	563
Satellite Radio Sample	568

Appendix E **Nike + iPod Cardio Equipment System 569**

Using Nike + iPod Cardio Equipment	569
Cardio Equipment Design	569
Determining Apple Device Support for Cardio Equipment	569
Identification Procedure	570
Declaring Cardio Equipment Support to the Apple Device	572
Accessing User Data	573
Recording Workout Data	573
XML Element Formats	576
User Interface Requirements	581
The Apple Device Does Not Support Nike + iPod	581
Deciding to Record a Workout to the Apple Device	581
The Apple Device is Full	581
User Weight	581
Recording Indicator	582
Workout Completed	582
Sample Command Sequence	584

Appendix F **Historical Information 591**

Past Protocol Features	591
The 3G iPod	595
General Compatibility With Accessories	595
Accessory Identification and iAP Commands	595
User Interface Restrictions	596
9-Pin Audio/Remote Connector Commands	596
Command 0x00: BeginRecord	596
Command 0x01: EndRecord	597
Command 0x02: BeginPlayback	597
Command 0x03: EndPlayback	598

General Lingo Command 0x01: Identify (Deprecated) 598

Deprecated Lingo 0x01: Microphone Lingo 600

 Command History of the Microphone Lingo 603

 Command 0x04: ACK 603

 Command 0x05: GetDevAck 604

 Command 0x06: iPodModeChange 605

 Command 0x07: GetDevCaps 606

 Command 0x08: RetDevCaps 607

 Command 0x09: GetDevCtrl 608

 Command 0x0A: RetDevCtrl 609

 Command 0x0B: SetDevCtrl 610

 Deprecated MicrophoneCapsToken 611

Deprecated Simple Remote Lingo Command 611

 Command 0x02: ImageButtonStatus 611

Deprecated USB Host Control Commands 613

 Command 0x00: ACK 613

 Command 0x01: GetUSBPowerState 613

 Command 0x02: RetUSBPowerState 614

 Command 0x03: SetUSBPowerState 614

Deprecated Extended Interface Commands 615

 Command 0x0012: RequestProtocolVersion 615

 Command 0x0013: ReturnProtocolVersion 616

 Command 0x0014: RequestiPodName 616

 Command 0x0015: ReturniPodName 617

 Command 0x0038: SelectSortDBRecord 618

Authentication 1.0 Sample Command Sequence 619

Accessory Identification With Non-IDPS Apple Devices 620

Glossary 643

Document Revision History 645

Figures, Tables, and Listings

Introduction **Introduction 31**

Table I-1 Apple device models 32

Chapter 1 **Protocol Features and Availability 41**

Figure 1-1 Sample iPod Out displays 60
Table 1-1 Current firmware and OS versions in Apple devices 41
Table 1-2 Most recent lingo versions for Apple devices, Table 1 42
Table 1-3 Most recent lingo versions for Apple devices, Table 2 43
Table 1-4 Most recent lingo versions for Apple devices, Table 3 43
Table 1-5 Features supported by specific Apple device firmware and OS versions, Table 1 44
Table 1-6 Features supported by specific Apple device firmware and OS versions, Table 2 45
Table 1-7 Features supported by specific Apple device firmware and OS versions, Table 3 46
Table 1-8 Video output capabilities 48
Table 1-9 Accessory communication commands 53
Table 1-10 USB Host mode commands 55
Table 1-11 Sample USB data transfer from accessory to host 57
Table 1-12 Video preferences for iPod Out mode 61
Table 1-13 Simple Remote and Display Remote commands supported in iPod Out mode 62

Chapter 2 **The Protocol Core and the General Lingo 65**

Figure 2-1 Screen configuration examples 124
Table 2-1 Command traffic for UART accessory identification 67
Table 2-2 Commands for accessory identification via USB or BT 69
Table 2-3 Apple device authentication coprocessor classes 72
Table 2-4 Lingo commands requiring authentication 72
Table 2-5 Command traffic for accessory authentication 74
Table 2-6 Accessory authentication of the Apple Device 76
Table 2-7 Small packet format 77
Table 2-8 Large packet format 77
Table 2-9 General lingo commands 79
Table 2-10 General lingo revision history 82
Table 2-11 RequestIdentify packet 83
Table 2-12 ACK packet 84
Table 2-13 ACK packet with transaction IDs 84
Table 2-14 ACK command error codes 85
Table 2-15 ACK packet with Command Pending status but no transaction ID 86

Table 2-16	ACK packet with Command Pending status and transaction ID	86
Table 2-17	ACK packet reporting dropped data	87
Table 2-18	RequestRemoteUIMode packet	88
Table 2-19	ReturnRemoteUIMode packet	88
Table 2-20	EnterRemoteUIMode packet	89
Table 2-21	ExitRemoteUIMode packet	90
Table 2-22	RequestiPodName packet	90
Table 2-23	ReturniPodName packet	91
Table 2-24	RequestiPodSoftwareVersion packet	92
Table 2-25	ReturniPodSoftwareVersion packet	92
Table 2-26	RequestiPodSerialNum packet	93
Table 2-27	ReturniPodSerialNum packet	93
Table 2-28	RequestLingoProtocolVersion packet	94
Table 2-29	ReturnLingoProtocolVersion packet	94
Table 2-30	RequestTransportMaxPayloadSize packet	95
Table 2-31	ReturnTransportMaxPayloadSize packet	96
Table 2-32	IdentifyDeviceLingoes packet	97
Table 2-33	Device Lingoes Spoken bits	98
Table 2-34	IdentifyDeviceLingoes Options bits	98
Table 2-35	GetDevAuthenticationInfo packet	100
Table 2-36	RetDevAuthenticationInfo packet, Authentication 1.0	100
Table 2-37	RetDevAuthenticationInfo packet, Authentication 2.0	101
Table 2-38	AckDevAuthenticationInfo packet	102
Table 2-39	GetDevAuthenticationSignature packet	103
Table 2-40	RetDevAuthenticationSignature packet	104
Table 2-41	AckDevAuthenticationStatus packet	104
Table 2-42	GetiPodAuthenticationInfo packet	105
Table 2-43	RetiPodAuthenticationInfo packet	105
Table 2-44	AckiPodAuthenticationInfo packet	106
Table 2-45	GetiPodAuthenticationSignature packet	107
Table 2-46	RetiPodAuthenticationSignature packet	108
Table 2-47	AckiPodAuthenticationStatus packet	108
Table 2-48	NotifyiPodStateChange packet	109
Table 2-49	GetiPodOptions packet	110
Table 2-50	RetiPodOptions packet	110
Table 2-51	Accessory Info Type values	112
Table 2-52	GetAccessoryInfo packet with Accessory Info Type = 0x00-0x01, 0x04-0x09, or 0x0B	112
Table 2-53	GetAccessoryInfo packet with Accessory Info Type = 0x02 (deprecated)	113
Table 2-54	GetAccessoryInfo packet with Accessory Info Type = 0x03	113
Table 2-55	Accessory Info Type values for RetAccessoryInfo	114
Table 2-56	RetAccessoryInfo packet with Accessory Info Type = 0x00	115
Table 2-57	Accessory info capabilities bit field	115
Table 2-58	RetAccessoryInfo packet with Accessory Info Type = 0x01/0x06/0x07/0x08	116
Table 2-59	RetAccessoryInfo packet with Accessory Info Type = 0x02 (deprecated)	117
Table 2-60	RetAccessoryInfo packet with Accessory Info Type = 0x03	118

Table 2-61	RetAccessoryInfo packet with Accessory Info Type = 0x04/0x05	118
Table 2-62	RetAccessoryInfo packet with Accessory Info Type = 0x09	119
Table 2-63	RetAccessoryInfo packet with Accessory Info Type = 0x0B	119
Table 2-64	Accessory status values	120
Table 2-65	GetiPodPreferences packet	120
Table 2-66	Apple device preference class and setting IDs	120
Table 2-67	RetiPodPreferences packet	125
Table 2-68	SetiPodPreferences packet	126
Table 2-69	GetUIMode packet	127
Table 2-70	RetUIMode packet	127
Table 2-71	User interface mode values	128
Table 2-72	SetUIMode packet	128
Table 2-73	StartIDPS packet	129
Table 2-74	SetFIDTokenValues packet	130
Table 2-75	FIDTokenValues field format	131
Table 2-76	FIDTokenValues tokens	131
Table 2-77	IdentifyToken format	132
Table 2-78	DeviceOptions bits in IdentifyToken	133
Table 2-79	AccCapsToken format	133
Table 2-80	Accessory capabilities bit values	133
Table 2-81	AccInfoToken format	135
Table 2-82	Accessory Info Type values	135
Table 2-83	Accessory RF certification declarations	136
Table 2-84	iPodPreferenceToken format	136
Table 2-85	EAProTOCOLToken format	136
Table 2-86	BundleSeedIDPrefToken format	137
Table 2-87	EAProTOCOLMetadataToken format	137
Table 2-88	MetadataType values	137
Table 2-89	ScreenInfoToken format	138
Table 2-90	Accessory screen features	139
Table 2-91	RetFIDTokenValueACKs packet	139
Table 2-92	Acknowledgment format for IdentifyToken	139
Table 2-93	Acknowledgment status codes	140
Table 2-94	Acknowledgment format for AccCapsToken	140
Table 2-95	Acknowledgment format for AccInfoToken	140
Table 2-96	Acknowledgment format for iPodPreferenceToken	141
Table 2-97	Acknowledgment format for EAProTOCOLToken	141
Table 2-98	Acknowledgment format for BundleSeedIDPrefToken	141
Table 2-99	Acknowledgment format for EAProTOCOLMetadataToken	142
Table 2-100	Acknowledgment format for MicCapsToken	142
Table 2-101	Acknowledgment format for ScreenInfoToken	142
Table 2-102	Acknowledgment status codes	142
Table 2-103	EndIDPS packet	143
Table 2-104	accEndIDPSStatus values	143
Table 2-105	IDPSStatus packet	144
Table 2-106	Apple device actions in response to accEndIDPSStatus and IDPS status	145

Table 2-107	OpenDataSessionForProtocol packet	146
Table 2-108	CloseDataSession packet	147
Table 2-109	DevACK packet	147
Table 2-110	DevDataTransfer small packet	149
Table 2-111	DevDataTransfer large packet	149
Table 2-112	iPodDataTransfer small packet	151
Table 2-113	iPodDataTransfer large packet	151
Table 2-114	SetAccStatusNotification packet	152
Table 2-115	RetAccStatusNotification packet	153
Table 2-116	AccessoryStatusNotification packet	154
Table 2-117	Accessory status types	155
Table 2-118	Accessory status parameters (fault)	155
Table 2-119	Accessory fault types	155
Table 2-120	Accessory fault conditions	155
Table 2-121	Sample IDPS process and accessory status notification commands	155
Table 2-122	SetEventNotification packet	157
Table 2-123	Notification bitmask bits	158
Table 2-124	iPodNotification packet for Flow Control Notifications	160
Table 2-125	iPodNotification packet for Radio Tagging Notifications	161
Table 2-126	iPodNotification payload for Radio Tagging notifications	161
Table 2-127	iPodNotification packet for Camera notifications	162
Table 2-128	iPodNotification payload for Camera Notifications	162
Table 2-129	iPodNotification packet for NowPlayingFocusApp notifications	162
Table 2-130	iPodNotification packet for Session Space Available notifications	163
Table 2-131	iPodNotification packet for Database Available notifications	163
Table 2-132	iPodNotification packet for Command Complete notifications	164
Table 2-133	iPodNotification packet for USB Host mode charging notifications	164
Table 2-134	InfoType values for USB Host mode charging notifications	165
Table 2-135	Charging Info values for iPodNotification	165
Table 2-136	GetiPodOptionsForLingo packet	166
Table 2-137	RetiPodOptionsForLingo packet	166
Table 2-138	RetiPodOptionsForLingo option bits	167
Table 2-139	Sample GetiPodOptionsForLingo and RetiPodOptionsForLingo commands	170
Table 2-140	GetEventNotification packet	176
Table 2-141	RetEventNotification packet	177
Table 2-142	GetSupportedEventNotification packet	177
Table 2-143	CancelCommand packet	178
Table 2-144	RetSupportedEventNotification packet	179
Table 2-145	SetAvailableCurrent packet	180
Table 2-146	RequestApplicationLaunch packet	181
Table 2-147	GetNowPlayingFocusApp packet	182
Table 2-148	RetNowPlayingFocusApp packet	182

Chapter 3

Accessory Lingoes 183

Figure 3-1 Typical digital audio transactions between an Apple device and an accessory 353

Figure 3-2	A USB host recovers from a CRC16 error	356
Table 3-1	iPod accessory lingo	183
Table 3-2	Select Apple device command timings	184
Table 3-3	Simple remote lingo command summary	185
Table 3-4	Simple remote lingo support versions	187
Table 3-5	Simple Remote lingo command history	187
Table 3-6	5G nano camera modes and transitions	190
Table 3-7	Suggested accessory feedback indications	191
Table 3-8	iPod camera interface commands	192
Table 3-9	Sample video control commands, accessory on and off	193
Table 3-10	Sample video control commands, iPod on and accessory off	196
Table 3-11	USB HID commands	197
Table 3-12	USB HID Consumer Page controls supported by iOS	197
Table 3-13	Simple Remote lingo accessibility commands	198
Table 3-14	ContextButtonStatus packet	198
Table 3-15	Button states	199
Table 3-16	ACK packet	200
Table 3-17	Command status codes	201
Table 3-18	VideoButtonStatus packet	202
Table 3-19	Video-specific button values	202
Table 3-20	AudioButtonStatus packet	203
Table 3-21	Audio-specific button values	204
Table 3-22	iPodOutButtonStatus packet	205
Table 3-23	iPod Out control locations	206
Table 3-24	iPod Out button status values	206
Table 3-25	RotationInputStatus packet	207
Table 3-26	Wheel types	208
Table 3-27	Rotation directions	208
Table 3-28	Rotation actions	208
Table 3-29	Rotation types	208
Table 3-30	RadioButtonStatus packet	211
Table 3-31	CameraButtonStatus packet	212
Table 3-32	RegisterDescriptor packet	212
Table 3-33	Country codes	213
Table 3-34	SendHIDReportToiPod packet	215
Table 3-35	SendHIDReportToAcc packet	216
Table 3-36	UnregisterDescriptor packet	216
Table 3-37	AccessibilityEvent packet	217
Table 3-38	AccessibilityEvent events and data	217
Table 3-39	GetAccessibilityParameter packet	219
Table 3-40	RetAccessibilityParameter packet	220
Table 3-41	Accessibility parameter types and values	220
Table 3-42	SetAccessibilityParameter packet	221
Table 3-43	GetCurrentItemProperty packet	222
Table 3-44	RetCurrentItemProperty packet	222
Table 3-45	Currently selected item property types and values	223

Table 3-46	SetContext packet	224
Table 3-47	User interface contexts	225
Table 3-48	AccParameterChanged packet	225
Table 3-49	DevACK packet	226
Table 3-50	Display Remote lingo command summary	227
Table 3-51	Display Remote lingo command history	229
Table 3-52	ACK packet	230
Table 3-53	Command result values	231
Table 3-54	GetCurrentEQProfileIndex packet	231
Table 3-55	RetCurrentEQProfileIndex packet	232
Table 3-56	SetCurrentEQProfileIndex packet	232
Table 3-57	GetNumEQProfiles packet	233
Table 3-58	RetNumEQProfiles packet	234
Table 3-59	GetIndexedEQProfileName packet	234
Table 3-60	RetIndexedEQProfileName packet	235
Table 3-61	SetRemoteEventNotification packet	236
Table 3-62	Apple device events	236
Table 3-63	RemoteEventNotification packet	238
Table 3-64	Event notification data	238
Table 3-65	Play status values	243
Table 3-66	Shuffle state	243
Table 3-67	Repeat state	243
Table 3-68	Power and battery state	244
Table 3-69	Audiobook playback speeds	244
Table 3-70	GetRemoteEventStatus packet	245
Table 3-71	RetRemoteEventStatus packet	245
Table 3-72	GetiPodStateInfo packet	246
Table 3-73	infoType values	246
Table 3-74	GetiPodStateInfo packet to retrieve the Apple device Equalizer Setting	247
Table 3-75	RetiPodStateInfo packet	248
Table 3-76	RetiPodStateInfo packet for requesting chapter information	248
Table 3-77	SetiPodStateInfo packet	249
Table 3-78	Apple device state data	250
Table 3-79	Restore-on-exit values	253
Table 3-80	SetiPodStateInfo packet for setting the current track	253
Table 3-81	GetPlayStatus packet	254
Table 3-82	RetPlayStatus packet	255
Table 3-83	SetCurrentPlayingTrack packet	256
Table 3-84	GetIndexedPlayingTrackInfo packet	256
Table 3-85	RetIndexedPlayingTrackInfo packet	257
Table 3-86	Track information data	258
Table 3-87	GetNumPlayingTracks packet	260
Table 3-88	RetNumPlayingTracks packet	260
Table 3-89	GetArtworkFormats packet	261
Table 3-90	RetArtworkFormats packet	261
Table 3-91	Display pixel format codes	262

Table 3-92	GetTrackArtworkData packet	262
Table 3-93	RetTrackArtworkData packet	263
Table 3-94	GetPowerBatteryState packet	265
Table 3-95	RetPowerBatteryState packet	265
Table 3-96	GetSoundCheckState packet	266
Table 3-97	RetSoundCheckState packet	266
Table 3-98	SetSoundCheckState packet	267
Table 3-99	GetTrackArtworkTimes packet	267
Table 3-100	RetTrackArtworkTimes packet	268
Table 3-101	CreateGeniusPlaylist command	269
Table 3-102	ACK responses to CreateGeniusPlaylist	270
Table 3-103	IsGeniusAvailableForTrack command	270
Table 3-104	ACK responses to IsGeniusAvailableForTrack	271
Table 3-105	Accessory Power lingo command summary	272
Table 3-106	Accessory Power lingo command history	272
Table 3-107	BeginHighPower packet	272
Table 3-108	EndHighPower packet	273
Table 3-109	USB Host mode commands	273
Table 3-110	Command history of the USB Host Control and Host Mode lingoes	274
Table 3-111	DevACK packet	274
Table 3-112	DevACK command error codes	275
Table 3-113	NotifyUSBMode packet	275
Table 3-114	USB Host mode status codes	276
Table 3-115	iPodACK packet	276
Table 3-116	iPodACK command error codes	276
Table 3-117	GetiPodUSBMode packet	277
Table 3-118	RetiPodUSBMode packet	277
Table 3-119	SetiPodUSBMode packet	278
Table 3-120	SetiPodUSBMode codes	278
Table 3-121	RF Tuner lingo command summary	280
Table 3-122	RF Tuner lingo command history	282
Table 3-123	RF tuner lingo ACK packet with cmdStatus not 0x06	283
Table 3-124	RF tuner lingo ACK packet with cmdStatus equal to 0x06	283
Table 3-125	RF tuner lingo ACK command status values	284
Table 3-126	GetTunerCaps packet	284
Table 3-127	RetTunerCaps packet	285
Table 3-128	RF tuner accessory capabilities payload	285
Table 3-129	Minimum FM resolution ID bits	286
Table 3-130	GetTunerCtrl packet	287
Table 3-131	RetTunerCtrl packet	287
Table 3-132	Tuner control state bits	288
Table 3-133	SetTunerCtrl packet	289
Table 3-134	Tuner control bits	289
Table 3-135	GetTunerBand packet	290
Table 3-136	RetTunerBand packet	290
Table 3-137	Tuner band state IDs	290

Table 3-138	SetTunerBand packet	291
Table 3-139	GetTunerFreq packet	292
Table 3-140	RetTunerFreq packet	292
Table 3-141	SetTunerFreq packet	293
Table 3-142	GetTunerMode packet	294
Table 3-143	RetTunerMode packet	294
Table 3-144	RF Tuner mode status bits	295
Table 3-145	SetTunerMode packet	295
Table 3-146	Set RF Tuner mode bits	295
Table 3-147	GetTunerSeekRssi packet	296
Table 3-148	RetTunerSeekRssi packet	297
Table 3-149	SetTunerSeekRssi packet	297
Table 3-150	TunerSeekStart packet	298
Table 3-151	Tuner seeking operations	298
Table 3-152	TunerSeekDone packet	300
Table 3-153	GetTunerStatus packet	301
Table 3-154	RetTunerStatus packet	301
Table 3-155	RF tuner status bits	301
Table 3-156	GetStatusNotifyMask packet	302
Table 3-157	RetStatusNotifyMask packet	303
Table 3-158	Status notification mask bits	303
Table 3-159	SetStatusNotifyMask packet	304
Table 3-160	Status notification mask setting bits	304
Table 3-161	StatusChangeNotify packet	305
Table 3-162	Status change ID bits	305
Table 3-163	GetRdsReadyStatus packet	306
Table 3-164	RetRdsReadyStatus packet	306
Table 3-165	RDS/RBDS data-ready status bits, parsed mode	307
Table 3-166	RDS/RBDS data-ready status bits, raw mode	307
Table 3-167	GetRdsData packet	307
Table 3-168	RDS/RBDS data type IDs, parsed mode	308
Table 3-169	RDS/RBDS data type IDs, raw mode	308
Table 3-170	RetRdsData packet	308
Table 3-171	rdsDataType bytes and rdsData formats	309
Table 3-172	rdsData character set IDs	309
Table 3-173	RDS/RBDS group data-ready data	310
Table 3-174	Block errors byte encoding	310
Table 3-175	Block error bit values	310
Table 3-176	GetRdsNotifyMask packet	311
Table 3-177	RetRdsNotifyMask packet	311
Table 3-178	RDS/RBDS data change notification mask bits, parsed mode	312
Table 3-179	RDS/RBDS data change notification mask bits, raw mode	312
Table 3-180	SetRdsNotifyMask packet	312
Table 3-181	RDS/RBDS data change notification mask setting bits, parsed mode	313
Table 3-182	RDS/RBDS data change notification mask setting bits, raw mode	313
Table 3-183	RdsReadyNotify packet	313

Table 3-184	GetHdProgramServiceCount packet	314
Table 3-185	RetHdProgramServiceCount packet	315
Table 3-186	GetHdProgramService packet	315
Table 3-187	RetHdProgramService packet	316
Table 3-188	SetHdProgramService packet	316
Table 3-189	GetHddDataReadyStatus packet	317
Table 3-190	RetHddDataReadyStatus packet	317
Table 3-191	HD data-ready status bits	318
Table 3-192	GetHddData packet	319
Table 3-193	HD data type IDs	319
Table 3-194	RetHddData packet	320
Table 3-195	HddDataType type IDs and formats	320
Table 3-196	PSD data format	321
Table 3-197	8-bit character encoding type formats	321
Table 3-198	GetHddDataNotifyMask packet	321
Table 3-199	RetHddDataNotifyMask packet	322
Table 3-200	HD data change notification mask bits	322
Table 3-201	SetHddDataNotifyMask packet	323
Table 3-202	HD data change notification mask setting bits	323
Table 3-203	HddDataReadyNotify packet	324
Table 3-204	HD data types	324
Table 3-205	HD Data Type type IDs and formats	325
Table 3-206	PSD data format	326
Table 3-207	8-bit character encoding type formats	326
Table 3-208	Example of HD radio setup	326
Table 3-209	Example of HD radio tuning and reception	327
Table 3-210	Example of getting PSD and name data	329
Table 3-211	Accessory Equalizer lingo command summary	330
Table 3-212	Accessory Equalizer lingo command history	331
Table 3-213	ACK packet	331
Table 3-214	GetCurrentEQIndex packet	332
Table 3-215	RetCurrentEQIndex packet	332
Table 3-216	Accessory Equalizer Setting indices	333
Table 3-217	SetCurrentEQIndex packet	333
Table 3-218	GetEQSettingCount packet	334
Table 3-219	RetEQSettingCount packet	334
Table 3-220	RetEQSettingCount parameter values	334
Table 3-221	GetEQIndexName packet	335
Table 3-222	RetEQIndexName packet	335
Table 3-223	Sports lingo command summary	336
Table 3-224	Sports lingo command history	337
Table 3-225	DeviceACK packet	337
Table 3-226	ackStatus details	338
Table 3-227	GetDeviceVersion packet	338
Table 3-228	RetDeviceVersion packet	339
Table 3-229	GetDeviceCaps packet	340

Table 3-230	RetDeviceCaps packet	340
Table 3-231	RetDeviceCaps capsMask details	341
Table 3-232	iPodACK packet	341
Table 3-233	iPodACK ackStatus details	341
Table 3-234	GetiPodCaps packet	342
Table 3-235	RetiPodCaps packet	342
Table 3-236	RetiPodCaps capsMask details	343
Table 3-237	GetUserIndex packet	343
Table 3-238	RetUserIndex packet	344
Table 3-239	GetUserData packet	345
Table 3-240	GetUserData userDataType details	345
Table 3-241	RetUserData packet	346
Table 3-242	RetUserData userDataType details	346
Table 3-243	SetUserData packet	347
Table 3-244	SetUserData userDataType details	348
Table 3-245	Digital Audio lingo command summary	349
Table 3-246	Digital Audio lingo command history	350
Table 3-247	AccAck packet	356
Table 3-248	AccAck status values	357
Table 3-249	iPodAck packet	357
Table 3-250	GetAccSampleRateCaps packet	358
Table 3-251	RetAccSampleRateCaps packet	359
Table 3-252	Digital audio sample rates supported by Apple devices (in Hertz)	359
Table 3-253	TrackNewAudioAttributes packet	360
Table 3-254	SetVideoDelay packet	361
Table 3-255	Storage lingo command history	362
Table 3-256	Storage lingo commands	362
Table 3-257	Storage iPodACK packet	364
Table 3-258	iPodACK responses	364
Table 3-259	GetiPodCaps packet	365
Table 3-260	RetiPodCaps packet	365
Table 3-261	RetiPodFileHandle packet	367
Table 3-262	WriteiPodFileData packet	368
Table 3-263	CloseiPodFile packet	368
Table 3-264	GetiPodFreeSpace packet	369
Table 3-265	RetiPodFreeSpace packet	369
Table 3-266	OpeniPodFeatureFile packet	370
Table 3-267	featureType values	371
Table 3-268	fileOptionsMask values	371
Table 3-269	DeviceACK packet	372
Table 3-270	ackStatus values	372
Table 3-271	GetDeviceCaps packet	373
Table 3-272	RetDeviceCaps packet	373
Table 3-273	iPod Out lingo command history	374
Table 3-274	iPod Out lingo command summary	374
Table 3-275	iPodACK packet	375

Table 3-276	iPodACK command status values	375
Table 3-277	GetiPodOutOptions packet	376
Table 3-278	RetiPodOutOptions packet	377
Table 3-279	Options for the iPod Out output from the Apple device	377
Table 3-280	SetiPodOutOptions packet	378
Table 3-281	DevStateChangeEvent packet	378
Table 3-282	Accessory state transitions	379
Table 3-283	DevVideoScreenInfo packet	379
Table 3-284	Location lingo commands	382
Table 3-285	Sample command interchange	383
Table 3-286	Default DevACK packet	386
Table 3-287	Valid Location lingo ackStatus values	386
Table 3-288	GetDevCaps packet	387
Table 3-289	Values for locType	387
Table 3-290	RetDevCaps packet	388
Table 3-291	System capabilities values (locType = 0x00)	388
Table 3-292	NMEA GPS location capabilities values (locType = 0x01)	389
Table 3-293	Location assistance capabilities values (locType = 0x02)	390
Table 3-294	GetDevControl packet	390
Table 3-295	RetDevControl packet	391
Table 3-296	SetDevControl packet	392
Table 3-297	ctlData values	393
Table 3-298	GetDevData packet	394
Table 3-299	GetDevData dataType values	394
Table 3-300	RetDevData packet	395
Table 3-301	RetDevData locData values	396
Table 3-302	locAsstData values (locType = 0x02) for RetDevData	396
Table 3-303	SetDevData packet	397
Table 3-304	Data types settable by SetDevData	398
Table 3-305	nmeaGpsLocData values (locType = 0x01)	398
Table 3-306	locAsstData values (locType = 0x02) for SetDevData	398
Table 3-307	AsyncDevData packet	400
Table 3-308	locData values that can be sent by AsyncDevData	401
Table 3-309	Default iPodACK packet	402
Table 3-310	Identification of lingo 0x00+0x04	403
Table 3-311	Identification of lingo 0x00+0x02+0x03	404
Table 3-312	Identification of lingo 0x00+0x02+0x03+0x0A	406

Chapter 4**The Extended Interface Protocol 409**

Figure 4-1	Typical “OK to disconnect” screens	410
Figure 4-2	An example interaction between an Extended Interface accessory and an Apple device	424
Figure 4-3	Navigating to a TV show: example of interactions between an accessory and an Apple device	428

Figure 4-4	Navigating to a movie: example of interactions between an accessory and an Apple device 430
Table 4-1	Small packet format 411
Table 4-2	Large packet format 412
Table 4-3	RetiPodOptionsForLingo option bits for Lingo 0x04 413
Table 4-4	Simplified IDPS and authentication command sequence 413
Table 4-5	General lingo commands for switching modes 416
Table 4-6	Database category hierarchy (excluding podcasts) 419
Table 4-7	Database category hierarchy for podcasts 420
Table 4-8	Selecting playlists containing video 431
Table 4-9	Typical extended interface commands 432
Table 4-10	Extended Interface lingo command summary 436
Table 4-11	ACK command 440
Table 4-12	GetCurrentPlayingTrackChapterInfo command 441
Table 4-13	ReturnCurrentPlayingTrackChapterInfo command 441
Table 4-14	SetCurrentPlayingTrackChapter command 442
Table 4-15	GetCurrentPlayingTrackChapterPlayStatus command 443
Table 4-16	ReturnCurrentPlayingTrackChapterPlayStatus command 444
Table 4-17	GetCurrentPlayingTrackChapterName command 445
Table 4-18	ReturnCurrentPlayingTrackChapterName command 445
Table 4-19	GetAudiobookSpeed command 446
Table 4-20	ReturnAudiobookSpeed command 446
Table 4-21	Audiobook speed states 447
Table 4-22	SetAudiobookSpeed command to set the speed of the currently playing audiobook 448
Table 4-23	SetAudiobookSpeed command to set the global audiobook speed 448
Table 4-24	GetIndexedPlayingTrackInfo command 449
Table 4-25	Track information type 449
Table 4-26	ReturnIndexedPlayingTrackInfo command for info types 0x00-0x02 and 0x05-0x07 450
Table 4-27	ReturnIndexedPlayingTrackInfo command for info types 0x03-0x04 451
Table 4-28	Track info types and return data 452
Table 4-29	Track Capabilities and Information encoding 452
Table 4-30	Track Release Date encoding 453
Table 4-31	GetArtworkFormats packet 453
Table 4-32	RetArtworkFormats packet 454
Table 4-33	GetTrackArtworkData packet 455
Table 4-34	RetTrackArtworkData packet 456
Table 4-35	ResetDBSelection command 457
Table 4-36	SelectDBRecord command 459
Table 4-37	Database category types for commands 460
Table 4-38	GetNumberCategorizedDBRecords command 461
Table 4-39	ReturnNumberCategorizedDBRecords command 461
Table 4-40	RetrieveCategorizedDatabaseRecords command 462
Table 4-41	ReturnCategorizedDatabaseRecord command 463
Table 4-42	GetPlayStatus command 464

Table 4-43	ReturnPlayStatus command	465
Table 4-44	GetCurrentPlayingTrackIndex command	466
Table 4-45	ReturnCurrentPlayingTrackIndex command	466
Table 4-46	GetIndexedPlayingTrackTitle command	467
Table 4-47	ReturnIndexedPlayingTrackTitle command	468
Table 4-48	GetIndexedPlayingTrackArtistName command	468
Table 4-49	ReturnIndexedPlayingTrackArtistName command	469
Table 4-50	GetIndexedPlayingTrackAlbumName command	470
Table 4-51	ReturnIndexedPlayingTrackAlbumName command	471
Table 4-52	One-byte SetPlayStatusChangeNotification command	471
Table 4-53	Four-byte SetPlayStatusChangeNotification command	472
Table 4-54	One-byte status change event values	472
Table 4-55	Four-byte status change event mask bits	473
Table 4-56	PlayStatusChangeNotification command	473
Table 4-57	Play status change notification codes	474
Table 4-58	PlayCurrentSelection command	476
Table 4-59	PlayControl command	477
Table 4-60	Play control command codes	477
Table 4-61	GetTrackArtworkTimes packet	478
Table 4-62	RetTrackArtworkTimes packet	479
Table 4-63	GetShuffle command	480
Table 4-64	ReturnShuffle command	481
Table 4-65	Shuffle modes	481
Table 4-66	SetShuffle command with Restore on Exit byte	482
Table 4-67	SetShuffle command	482
Table 4-68	GetRepeat command	483
Table 4-69	ReturnRepeat command	483
Table 4-70	Repeat state values	484
Table 4-71	SetRepeat command with Restore on Exit byte	485
Table 4-72	SetRepeat command	485
Table 4-73	SetDisplayImage descriptor command (packet index = 0x0000)	486
Table 4-74	SetDisplayImage data packet (packet index = 0x0001 – 0xNNNN)	488
Table 4-75	Display pixel format codes	489
Table 4-76	2 bpp monochrome pixel intensities	489
Table 4-77	GetMonoDisplayImageLimits command	491
Table 4-78	ReturnMonoDisplayImageLimits command	492
Table 4-79	GetNumPlayingTracks command	493
Table 4-80	ReturnNumPlayingTracks command	493
Table 4-81	SetCurrentPlayingTrack command	494
Table 4-82	GetColorDisplayImageLimits command	495
Table 4-83	ReturnColorDisplayImageLimits command	495
Table 4-84	ResetDBSelectionHierarchy command	496
Table 4-85	GetDBiTunesInfo command	497
Table 4-86	iTunes database metadata types	497
Table 4-87	RetDBiTunesInfo command	498
Table 4-88	iTunes database metadata information formats	498

Table 4-89	Date/time format	499
Table 4-90	GetUIDTrackInfo command	499
Table 4-91	Track information type bits	500
Table 4-92	RetUIDTrackInfo command	502
Table 4-93	Track information data formats	502
Table 4-94	Capabilities bits	504
Table 4-95	GetDBTrackInfo command	505
Table 4-96	RetDBTrackInfo command	506
Table 4-97	GetPBTrackInfo command	507
Table 4-98	RetPBTrackInfo command	508
Table 4-99	CreateGeniusPlaylist command	508
Table 4-100	Index types	509
Table 4-101	ACK responses to CreateGeniusPlaylist	509
Table 4-102	RefreshGeniusPlaylist command	510
Table 4-103	ACK responses to RefreshGeniusPlaylist	510
Table 4-104	IsGeniusAvailableForTrack command	511
Table 4-105	ACK responses to IsGeniusAvailableForTrack	511
Table 4-106	GetPlaylistInfo command	512
Table 4-107	GetPlaylistInfo info types	512
Table 4-108	RetPlaylistInfo command	513
Table 4-109	History of the Extended Interface protocol	514

Appendix A **Accessory Identification** 519

Table A-1	IDPS General lingo commands	520
Table A-2	IDPS process up to authentication	521
Table A-3	IDPS process with authentication and Digital Audio lingo	522

Appendix B **Transaction IDs** 525

Table B-1	Forms of the General lingo ACK command	526
Table B-2	Example of IDPS and authentication	527
Table B-3	Example of entering Remote UI mode	528
Table B-4	Example of getting track artwork data	528
Table B-5	Example of sending notifications	529

Appendix C **Multisection Data Transfers** 531

Table C-1	Multisection iPodACK packet	532
Table C-2	Multisection DevACK packet	533

Appendix D **iTunes Tagging** 535

Figure D-1	iTunes tagging feature data flows	536
Figure D-2	iTunes tagging element sequence	552

Figure D-3	RDS 3A group for iTunes tagging	553
Figure D-4	RDS application group for iTunes tagging	554
Figure D-5	Event Control element	555
Figure D-6	Element decryption process	556
Table D-1	HD UFID data ID types	539
Table D-2	Typical plist writing command sequence	540
Table D-3	Plist fields written to the Apple device	541
Table D-4	Tagging feature user interface implementation	544
Table D-5	Tagging feature UI text messages	545
Table D-6	Radio tagging command sequence	546
Table D-7	Required plist fields for FM	550
Table D-8	RT+ content types	551
Table D-9	FM tag element types	552
Table D-10	Outer mask lookup table	556
Table D-11	Permutation lookup table, upper bits	557
Table D-12	Permutation lookup table, lower bits	558
Table D-13	Element type lookup table	558
Table D-14	Required plist fields for HD radio	560
Table D-15	Required plist fields for Satellite radio	561
Listing D-1	Pseudo-random number generator	559

Appendix E**Nike + iPod Cardio Equipment System 569**

Figure E-1	Sample user experience flow chart	583
Table E-1	Lingo version support	570
Table E-2	Sample identification process 1	571
Table E-3	Sample identification process 2	572
Table E-4	Writing workout data to the Apple device	573
Table E-5	XML element usage	577
Table E-6	Approved user interface messages	582
Table E-7	Cardio equipment command sequence using IDPS	584

Appendix F**Historical Information 591**

Table F-1	Past Apple device models, firmware, and lingo versions, table 1	591
Table F-2	Past Apple device models, firmware, and lingo versions, table 2	593
Table F-3	3G iPod product identification	595
Table F-4	BeginRecord packet	596
Table F-5	EndRecord packet	597
Table F-6	BeginPlayback packet	597
Table F-7	EndPlayback packet	598
Table F-8	Identify packet	599
Table F-9	Identify packet for high-power accessories	599
Table F-10	Power option bits	600
Table F-11	Microphone lingo command summary	602

Table F-12	Select Microphone lingo command timings	602
Table F-13	Microphone lingo command history	603
Table F-14	ACK packet	603
Table F-15	Command result values	604
Table F-16	GetDevAck packet	604
Table F-17	iPodModeChange packet	605
Table F-18	Mode values	606
Table F-19	GetDevCaps packet	607
Table F-20	RetDevCaps packet	607
Table F-21	Microphone capabilities bitmask	608
Table F-22	GetDevCtrl packet	608
Table F-23	RetDevCtrl packet	609
Table F-24	Control types and data	609
Table F-25	SetDevCtrl packet	610
Table F-26	MicrophoneCapsToken format	611
Table F-27	Microphone capabilities bits	611
Table F-28	ImageButtonStatus packet	612
Table F-29	Image-specific button values	612
Table F-30	ACK packet	613
Table F-31	GetUSBPowerState packet	614
Table F-32	RetUSBPowerState packet	614
Table F-33	SetUSBPowerState packet	615
Table F-34	RequestProtocolVersion command	615
Table F-35	ReturnProtocolVersion command	616
Table F-36	RequestiPodName command	617
Table F-37	ReturniPodName command	617
Table F-38	SelectSortDBRecord command	618
Table F-39	Database sort order options	619
Table F-40	Legacy accessory authentication	620
Table F-41	UART accessory identification with non-IDPS Apple devices	621
Table F-42	USB or BT accessory identification with non-IDPS Apple devices	622
Table F-43	Non-IDPS identification of lingo 0x00+0x04	623
Table F-44	Non-IDPS identification of lingo 0x00+0x02+0x03	628
Table F-45	Non-IDPS identification of lingo 0x00+0x02+0x03+0x0A	631
Table F-46	Non-IDPS radio tagging command sequence	633
Table F-47	Non-IDPS cardio equipment command sequence	636

Introduction

NOTICE OF PROPRIETARY PROPERTY: THE INFORMATION CONTAINED HEREIN IS THE PROPRIETARY PROPERTY OF APPLE INC. THE POSSESSOR AGREES TO THE FOLLOWING: (I) TO MAINTAIN THIS DOCUMENT IN CONFIDENCE, (II) NOT TO REPRODUCE OR COPY IT, (III) NOT TO REVEAL OR PUBLISH IT IN WHOLE OR IN PART, (IV) ALL RIGHTS RESERVED.

ACCESS TO THIS DOCUMENT AND THE INFORMATION CONTAINED THEREIN IS GOVERNED BY THE TERMS OF THE MFI LICENSE AGREEMENT AND/OR THE IPOD-IPHONE AIS EVALUATION AGREEMENT. ALL OTHER USE SHALL BE AT APPLE'S SOLE DISCRETION.

This document specifies the electrical and software interfaces to the Apple iPod, iPhone, and iPad that are available to accessories. The models covered by this specification are shown in [Table I-1](#) (page 32). This document does not apply to the first- and second-generation iPods, nor to the iPod shuffle.

IMPORTANT: This document uses the term “Apple device” to refer generically to iPods, iPhones, and iPads, all of which support the iPod Accessory Protocol (iAP) interface. Among these products, those that also run iOS (Apple's mobile operating system) are referred to as “iOS devices.” Specifications in this document that are designated for iOS devices apply only to those products. Specifications designated for iPods apply only to Apple devices that are not iOS devices.

To determine if a command or feature is supported in a particular Apple device model or firmware release, see [Table 1-5](#) (page 44), [Table 1-6](#) (page 45), and [Table 1-7](#) (page 46).



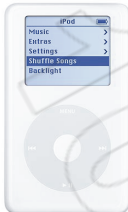







Note: All accessories for the iPhone must meet additional electrical and RF design requirements. These requirements are described in the iPhone license agreement and in Apple's *MFi Accessory Testing Specification*.

The minimum iPod firmware and iOS versions supported by this specification are:

- iPod firmware version 3.0 on the fourth-generation (4G) iPod.
- iPod firmware version 1.0 on the iPod mini, fourth-generation iPod (color display), iPod nano, fifth-generation iPod (5G), second-generation iPod nano, iPod classic, and iPod 3G nano.
- iPod firmware version 1.0.2 on the 4G nano.
- iOS version 1.1 on the iPhone, iPhone 3G, and iPod touch.
- iOS version 2.1.1 on the 2G iPod touch.
- iOS version 3.0.0 on the iPhone 3GS.
- iOS version 3.2.0 on the iPad.
- iOS version 4.0.0 on the iPhone 4.







Note: Supporting the 3G iPod requires special design considerations. See "The 3G iPod" (page 595) and "Interfacing With the 3G iPod" in *MFi Accessory Hardware Specification*.






Table I-1 Apple device models





iPod mini		
	<i>Product name:</i> iPod mini	<i>Shipped:</i> 01/2004
	<i>Compatibility icon:</i>  iPod mini 4GB 6GB	
	<i>Connectivity:</i> Dock connector, headphone jack	
4G iPod		
 	<i>Product names:</i> iPod (4th generation), iPod photo, iPod photo (2nd generation), iPod 4th generation (color display)	<i>Shipped:</i> 07/2004, 10/2004, 02/2005, 06/2005
	<i>Compatibility icons:</i>  iPod 4th generation 20GB  iPod 4th generation 40GB  iPod 4th generation (color display) 20GB 30GB  iPod 4th generation (color display) 40GB 60GB	
	<i>Connectivity:</i> Dock connector, headphone jack	
iPod nano		
	<i>Product name:</i> iPod nano	<i>Shipped:</i> 09/2005
	<i>Compatibility icon:</i>  iPod nano 1st generation 1GB 2GB 4GB	
	<i>Connectivity:</i> Dock connector, headphone jack	

5G iPod		
	<i>Product name:</i> iPod with video	<i>Shipped:</i> 10/2005
	<i>Compatibility icons:</i> <div><div><div>iPod 5th generation (video) 30GB</div></div><div><div>iPod 5th generation (video) 60GB 80GB</div></div></div>	
	<i>Connectivity:</i> Dock connector, headphone jack	
2G nano		
	<i>Product name:</i> iPod nano (2nd generation)	<i>Shipped:</i> 09/2006
	<i>Compatibility icon:</i> <div><div>iPod nano 2nd generation (aluminum) 2GB 4GB 8GB</div></div>	
	<i>Connectivity:</i> Dock connector, headphone jack	
iPod classic		
	<i>Product names:</i> iPod classic	<i>Shipped:</i> 09/2007
	<i>Compatibility icon:</i> <div><div><div>iPod classic 80GB</div></div><div><div>iPod classic 160GB (2007)</div></div></div>	
	<i>Connectivity:</i> Dock connector, headphone jack	
120 GB classic, 160 GB classic		
	<i>Product names:</i> iPod classic (120GB), iPod classic (160GB)	<i>Shipped:</i> 09/2008, 9/2009
	<i>Compatibility icon:</i> <div><div>iPod classic 160GB (2009)</div></div>	
	<i>Connectivity:</i> Dock connector, microphone/headphone jack	

3G nano		
	<i>Product name:</i> iPod nano (3rd generation)	<i>Shipped:</i> 09/2007
	<i>Compatibility icon:</i>  iPod nano 3rd generation (video) 4GB 8GB	
	<i>Connectivity:</i> Dock connector, headphone jack	
4G nano		
	<i>Product name:</i> iPod nano (4th generation)	<i>Shipped:</i> 09/2008
	<i>Compatibility icon:</i>  iPod nano 4th generation (video) 8GB 16GB	
	<i>Connectivity:</i> Dock connector, microphone/headphone jack	
iPhone		
	<i>Product name:</i> iPhone	<i>Shipped:</i> 06/2007
	<i>Compatibility icon:</i>  iPhone 4GB 8GB 16GB	
	<i>Connectivity:</i> GSM, Wi-Fi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	
iPod touch		
	<i>Product names:</i> iPod touch	<i>Shipped:</i> 09/2007
	<i>Compatibility icon:</i>  iPod touch 1st generation 8GB 16GB 32GB	
	<i>Connectivity:</i> Wi-Fi, multi-touch display, dock connector, headphone jack	

iPhone 3G		
	<i>Product name:</i> iPhone 3G	<i>Shipped:</i> 06/2008
	<i>Compatibility icon:</i>  iPhone 3G 8GB 16GB	
	<i>Connectivity:</i> GSM, Wi-Fi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	
2G touch		
	<i>Product names:</i> iPod touch (2nd generation)	<i>Shipped:</i> 09/2008
	<i>Compatibility icon:</i>  iPod touch 2nd generation 8GB 16GB 32GB	
	<i>Connectivity:</i> Wi-Fi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	
iPhone 3GS		
	<i>Product name:</i> iPhone 3GS	<i>Shipped:</i> 6/2009, 6/2010
	<i>Compatibility icon:</i>  iPhone 3GS 8GB 16GB 32GB	
	<i>Connectivity:</i> GSM, Wi-Fi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	
5G nano		
	<i>Product name:</i> iPod nano (5th generation)	<i>Shipped:</i> 09/2009
	<i>Compatibility icon:</i>  iPod nano 5th generation (video camera) 8GB 16GB	
	<i>Connectivity:</i> Dock connector, microphone/headphone jack	

3G touch		
	<i>Product names:</i> iPod touch (3rd generation)	<i>Shipped:</i> 09/2009
	<i>Compatibility icon:</i>  iPod touch 3rd generation 32GB 64GB	
	<i>Connectivity:</i> Wi-Fi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	
iPad		
	<i>Product names:</i> iPad (Wi-Fi), iPad (Wi-Fi + 3G)	<i>Shipped:</i> 04/2010
	<i>Compatibility icon:</i>  iPad 16GB 32GB 64GB	
	<i>Connectivity:</i> GSM (iPad 3G), Wi-Fi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	
iPhone 4		
	<i>Product names:</i> iPhone 4 (GSM model), iPhone 4 (CDMA model)	<i>Shipped:</i> 06/2010, 02/2011
	<i>Compatibility icon:</i>  iPhone 4 16GB 32GB	
	<i>Connectivity:</i> CDMA or GSM, Wi-Fi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	
6G nano		
	<i>Product name:</i> iPod nano (6th generation)	<i>Shipped:</i> 09/2010
	<i>Compatibility icon:</i>  iPod nano 6th generation 8GB 16GB	
	<i>Connectivity:</i> Multi-touch display, dock connector, microphone/headphone jack	

4G touch		
	<i>Product names:</i> iPod touch (4th generation)	<i>Shipped:</i> 09/2010
	<i>Compatibility icon:</i>  iPod touch 4th generation 8GB 32GB 64GB	
	<i>Connectivity:</i> Wi-Fi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	
iPad 2		
	<i>Product names:</i> iPad 2 (Wi-Fi), iPad 2 (Wi-Fi + 3G)	<i>Shipped:</i> 03/2011
	<i>Compatibility icon:</i>  iPad 2 16GB 32GB 64GB	
	<i>Connectivity:</i> CDMA or GSM (iPad 2 (WiFi + 3G)), Wi-Fi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	

Organization of This Document

The specifications in this document are arranged in four chapters:

- ["Protocol Features and Availability"](#) (page 41) gives an overview of the General and accessory lingo and their availability.
- ["The Protocol Core and the General Lingo"](#) (page 65) gives an overview of the iAP and describes the General Lingo, which all accessories must support.
- ["Accessory Lingo"](#) (page 183) describes the various accessory-specific lingo that are part of the iAP and specifies their commands.
- ["The Extended Interface Protocol"](#) (page 409) describes iAP Lingo 0x04, which allows the Apple device user interface to be translated to other environments.

Several appendixes provide additional information for both hardware and software designers:

- ["Accessory Identification"](#) (page 519) specifies the Identify Device Preferences and Settings (IDPS) process which accessories identify themselves.
- ["Transaction IDs"](#) (page 525) describes the transaction ID parameters used by the Location lingo, the IDPS process, and multisection data transfers.
- ["Multisection Data Transfers"](#) (page 531) tells how to transfer amounts of data larger than the data receiver's maximum incoming packet size.
- ["iTunes Tagging"](#) (page 535) describes the iTunes tagging feature for broadcast reception accessories.

- ["Nike + iPod Cardio Equipment System"](#) (page 569) describes the use of the Sports and Storage lingoes with Nike + iPod cardio equipment.
- ["Historical Information"](#) (page 591) provides specifications for past Apple devices and iAP protocols.

At the end of this document are a glossary of terms and a document revision history.

General Specification Terms

Parts of this document contain specification requirements that are incorporated by reference into legal agreements between Apple Inc. and its licensees. The use of the words "must," "should," and "may" in these specifications have the following meanings:

- "Must" means that the specification is an absolute requirement.
- "Must not" means that the specification is an absolute prohibition.
- "Should" means that there may be valid reasons in particular circumstances to ignore the specification, but their full implications must be understood and carefully weighed before choosing to do so.
- "Should not" means that there may be valid reasons in particular circumstances that make the specified action or feature acceptable, but their full implications must be understood and carefully weighed before choosing to include it.
- "May" means that the indicated action or feature does not contravene this specification.

Special Terminology

Certain terms in this document have the following specific meanings:

- When a data field is marked "Reserved," accessories writing to it must set it to 0 (unless otherwise noted) and accessories reading it must ignore its value.
- "Deprecated" marks an earlier technology that is no longer permitted in new accessory designs.
- "USB Host Mode" is an operating mode in which an Apple device is a USB host and its attached accessory acts as a USB device.
- "USB device mode" is an operating mode in which an accessory is a USB host and its attached Apple device acts as a USB device.

See Also

For further information, refer to the latest revisions of these additional documents:

- *IEEE 1394a Specification*
- *USB 2.0 High Speed Specification*

- *USB Device Class Definition for Audio Devices*
- *USB Device Class Definition for Audio Data Formats*
- *USB Device Class Definition for Human Interface Devices (HID)*
- *United States RBDS Standard, NRSC-4-A*

Protocol Features and Availability

[Table 1-1](#) (page 41) lists the current firmware and OS versions for each iPod, iPhone, and iPad model.

Table 1-1 Current firmware and OS versions in Apple devices

Model	Firmware version	Revision date	OS version	Revision date
iPod mini	1.4.1	01/2006		
4G iPod	3.1.1	01/2006		
4G iPod (color display)	1.2.1	01/2006		
1G nano	1.3.1	03/2007		
iPod 5G	1.3	03/2008		
2G nano	1.1.3	05/2007		
iPod classic	1.1.2	05/2008		
3G nano	1.1.3	07/2008		
iPod touch			iOS 3.1.3	02/2010
iPhone			iOS 3.1.3	02/2010
iPhone 3G			iOS 4.2.1	11/2010
4G nano	1.0.4	08/2009		
120 GB classic	2.0.1	01/2009		
2G touch			iOS 4.2.1	11/2010
iPhone 3GS			iOS 4.3.1	03/2011
160 GB classic	2.0.4	12/2009		
5G nano	1.0.2	11/2009		
iPad			iOS 4.3.1	03/2011
iPad 3G			iOS 4.3.1	03/2011
3G touch			iOS 4.3.1	03/2011
iPhone 4 (GSM model)			iOS 4.3.1	03/2011

Model	Firmware version	Revision date	OS version	Revision date
6G nano	1.0	09/2010		
4G touch			iOS 4.3.1	03/2011
iPhone 4 (CDMA model)			iOS 4.2.6	02/2011
iPad 2 (WiFi)			iOS 4.3.1	03/2011
iPad 2 (WiFi + 3G)			iOS 4.3.1	03/2011

Table 1-2 (page 42), Table 1-3 (page 43), and Table 1-4 (page 43) list the lingo protocol versions for each Apple device's most recent firmware and OS version. In these tables, "NL" indicates that the given lingo was not implemented in the specified model loaded with the specified firmware. "NV" indicates that although the lingo was implemented, its protocol version could not be read from the iPod.

Table 1-2 Most recent lingo versions for Apple devices, Table 1

Models	mini	4G	photo; 4G (color display)	1G nano	5G	2G nano	classic	3G nano	iPod touch
General 0x00	1.02	1.02	1.02	1.05	1.06	1.06	1.07	1.07	1.09
Simple Remote 0x02	1.00	1.00	1.00	1.02	1.02	1.02	1.02	1.02	1.02
Display Remote 0x03	1.01	1.01	1.01	1.05	1.05	1.04	1.05	1.05	1.05
Extended Interface 0x04	1.05	1.05	1.09	1.11	1.12	1.10	1.13	1.13	1.12
Accessory Power 0x05	1.00	1.00	1.00	1.01	1.01	1.01	1.01	1.01	1.01
USB Host Mode 0x06	NL	NL	1.00	NL	1.00	NL	NL	NL	NL
RF Tuner 0x07	NL	NL	NL	1.00	1.00	1.00	1.00	1.00	NL
Accessory Equalizer 0x08	NL	NL	NL	1.00	1.00	1.00	1.00	1.00	1.00
Sports 0x09	NL	NL	NL	NL	NL	NL	NL	1.01	NL
Digital Audio 0x0A	NL	NL	NL	1.01	1.01	1.02	1.03	1.03	1.02
Storage 0x0C	NL	NL	NL	NL	1.01	NL	1.02	1.02	1.02
Location 0x0E	NL	NL	NL	NL	NL	NL	NL	NL	1.00

Table 1-3 Most recent lingo versions for Apple devices, Table 2

Models	iPhone	iPhone 3G	4G nano	120 GB classic	2G, 3G touch	iPhone 3GS	160 GB classic	5G nano
General 0x00	1.09	1.09	1.08	1.08	1.09	1.09	1.08	1.09
Simple Remote 0x02	1.02	1.03	1.04	1.03	1.03	1.03	1.03	1.04
Display Remote 0x03	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05
Extended Interface 0x04	1.12	1.12	1.14	1.13	1.12	1.12	1.13	1.14
Accessory Power 0x05	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01
USB Host Mode 0x06	NL	NL	NL	NL	NL	NL	NL	NL
RF Tuner 0x07	NL	NL	1.01	1.00	NL	NL	1.00	1.01
Accessory Equalizer 0x08	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Sports 0x09	NL	NL	1.01	NL	1.01	1.01	NL	1.01
Digital Audio 0x0A	1.02	1.02	1.03	1.03	1.02	1.02	1.03	1.03
Storage 0x0C	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02
iPod Out 0x0D	NL	1.00	NL	NL	1.00	1.00	NL	NL
Location 0x0E	1.00	1.00	NL	NL	1.00	1.00	NL	NL

Table 1-4 Most recent lingo versions for Apple devices, Table 3

Models	iPad, iPad 3G, iPad 2 (WiFi), iPad 2 (WiFi + 3G)	iPhone 4 (GSM model), iPhone 4 (CDMA model)	6G nano	4G touch
General 0x00	1.09	1.09	1.09	1.09
Simple Remote 0x02	1.02	1.03	1.07	1.03
Display Remote 0x03	1.05	1.05	1.05	1.05
Extended Interface 0x04	1.12	1.12	1.14	1.12
Accessory Power 0x05	1.01	1.01	1.01	1.01
USB Host Mode 0x06	1.00	NL	1.00	NL
RF Tuner 0x07	NL	NL	1.01	NL

Models	iPad, iPad 3G, iPad 2 (WiFi), iPad 2 (WiFi + 3G)	iPhone 4 (GSM model), iPhone 4 (CDMA model)	6G nano	4G touch
Accessory Equalizer 0x08	1.00	1.00	1.00	1.00
Sports 0x09	NL	1.01	1.01	1.01
Digital Audio 0x0A	1.02	1.02	1.03	1.02
Storage 0x0C	1.02	1.02	1.02	1.02
iPod Out 0x0D	1.00	1.00	1.00	1.00
Location 0x0E	1.00	1.00	NL	1.00

The accessory should determine what features an attached Apple device supports by sending the General lingo command 0x4B, `GetiPodOptionsForLingo`. If the Apple device does not support that command, the accessory must extract the lingo version number from the Apple device. This protocol version information can be used to determine which features the connected Apple device supports. See "[Command 0x0F: RequestLingoProtocolVersion](#)" (page 94) for information about the `RequestLingoProtocolVersion` command. For past firmware versions, see [Table F-1](#) (page 591). [Table 1-5](#) (page 44) lists hardware and software features outside these protocols.

General Apple Device Features

[Table 1-5](#) (page 44), [Table 1-6](#) (page 45), and [Table 1-7](#) (page 46) list Apple device hardware and software features that are not part of specific lingoes. The numbers in the tables show the firmware versions in which each of these features was introduced.

Table 1-5 Features supported by specific Apple device firmware and OS versions, Table 1

Features	Software versions										
	3G	mini	4G	4G (color)	nano	5G	2G nano	classic	3G nano	iPod touch	iPhone
Serial autobaud on framing errors	—	1.4.0	3.1	1.1	1.0	1.0	1.0	1.0	1.0	1.1	1.1
Display notification on unsupported iAP accessory attach	—	—	—	—	1.0	1.0	1.0	1.0	1.0	1.1	1.1
Apple device detects detach for all valid R_{ID} values	—	1.4.0	3.1	1.2	1.0	1.0	1.0	1.0	1.0	1.1	1.1
Authentication Version 1.00	—	—	—	—	1.0	1.0	1.0	1.0	1.0	1.1	1.1

Features	Software versions										
	3G	mini	4G	4G (color)	nano	5G	2G nano	classic	3G nano	iPod touch	iPhone
Authentication Version 2.00	—	—	—	—	1.2	1.2	1.0	1.0	1.0	1.1	1.1
USB Device Mode audio output	—	—	—	—	1.2	1.2	1.1.2	1.0	1.0	1.1	1.1
Video browsing ²	—	—	—	—	—	1.2	—	1.0.3	1.0.3	2.0	1.1
iTunes tagging	—	—	—	—	—	1.2.3 ¹	—	1.0	1.0	2.1	2.1
Headphone remote ³ and mic system	—	—	—	—	—	—	—	—	—	—	—
Nike + iPod cardio equipment	—	—	—	—	—	—	—	—	1.1.2	—	—
Communication with iOS applications	—	—	—	—	—	—	—	—	—	3.0	3.0
Analog audio input (recording)	—	—	—	—	—	1.0	1.0	1.0	1.0	3.0	2.2.1

¹ Accessories must not use the Storage lingo over USB with the 5G iPod. Only UART transport is supported.

Table 1-6 Features supported by specific Apple device firmware and OS versions, Table 2

Features	Software versions							
	iPhone 3G	4G nano	120 GB classic	2G touch	iPhone 3GS	3G touch	160 GB classic	5G nano
Serial autobaud on framing errors	2.1	1.0.2	2.0	2.1.1	3.0	3.1	2.0.3	1.0.1
Display notification on unsupported iAP accessory attach	2.1	1.0.2	2.0	2.1.1	3.0	3.1	2.0.3	1.0.1
Apple device detects detach for all valid R _{ID} values	2.1	1.0.2	2.0	2.1.1	3.0	3.1	2.0.3	1.0.1
Authentication Version 1.00	2.1	1.0.2	2.0	2.1.1	3.0	3.1	2.0.3	1.0.1
Authentication Version 2.00	2.1	1.0.2	2.0	2.1.1	3.0	3.1	2.0.3	1.0.1

Features	Software versions							
	iPhone 3G	4G nano	120 GB classic	2G touch	iPhone 3GS	3G touch	160 GB classic	5G nano
USB Device Mode audio output	2.1	1.0.2	2.0	2.1.1	3.0	3.1	2.0.3	1.0.1
Video browsing ²	2.1	1.0.2	2.0	2.1.1	3.0	3.1	2.0.3	1.0.1
iTunes tagging	2.1	1.0.2	2.0	2.1.1	3.0	3.1	2.0.3	1.0.1
Headphone remote ³ and mic system	—	1.0.2	2.0	2.1.1	3.0	3.1	2.0.3	1.0.1
Nike + iPod cardio equipment	—	1.0.2	—	2.1.1	3.0	3.1	—	1.0.1
Communication with iOS applications	3.0	—	—	3.0	3.0	3.1	—	—
USB Host Mode ⁴	—	—	—	—	4.0	4.0	—	—
USB Host Mode audio input	—	—	—	—	—	—	—	—
Analog audio input (recording)	2.1	1.0.2	2.0	2.1.1	3.0	3.1	2.0.4	1.0.2
iPod Out	4.0	—	—	4.0	4.0	4.0	—	—

Table 1-7 Features supported by specific Apple device firmware and OS versions, Table 3

Features	Software versions					
	iPad, iPad 3G	iPhone 4 (GSM model)	6G nano	4G touch	iPhone 4 (CDMA model)	iPad 2 (WiFi), iPad 2 (WiFi + 3G)
Serial autobaud on framing errors	3.2	4.0	1.0	4.1	4.2.5	4.3
Display notification on unsupported iAP accessory attach	3.2	4.0	1.0	4.1	4.2.5	4.3
Apple device detects detach for all valid R _{ID} values	3.2	4.0	1.0	4.1	4.2.5	4.3
Authentication Version 1.00	3.2	4.0	1.0	4.1	4.2.5	4.3
Authentication Version 2.00	3.2	4.0	1.0	4.1	4.2.5	4.3

Features	Software versions					
	iPad, iPad 3G	iPhone 4 (GSM model)	6G nano	4G touch	iPhone 4 (CDMA model)	iPad 2 (WiFi), iPad 2 (WiFi + 3G)
USB Device Mode audio output	3.2	4.0	1.0	4.1	4.2.5	4.3
Video browsing ²	—	4.0	—	4.1	4.2.5	4.3
iTunes tagging	3.2	4.0	1.0	4.1	4.2.5	4.3
Headphone remote ³ and mic system	3.2	4.0	1.0	4.1	4.2.5	4.3
Nike + iPod cardio equipment	—	4.0	1.0	4.1	4.2.5	4.3
Communication with iOS applications	3.2	4.0	—	4.1	4.2.5	4.3
USB Host Mode ⁴	3.2	4.0	1.0	4.1	4.2.5	4.3
USB Host Mode audio input	3.2	4.0	—	4.1	4.2.5	4.3
Analog audio input (recording)	—	—	—	—	4.2.5	4.3
iPod Out	4.2.1	4.0	1.0	4.1	4.2.5	4.3

¹ Because version 1.00 of Lingo 0x0A contained a bug that was corrected in version 1.0.1, accessories should attempt Digital Audio only with Apple devices whose Lingo 0x0A version is higher than or equal to 1.0.1. See [Table 3-246](#) (page 350).

² With Apple devices that store and display video content (in addition to music), an accessory may choose to navigate the Apple device's hierarchy of stored videos. For details, see ["Video Browsing"](#) (page 426).

³ In the headphone remote and mic system, remote control from the headphone is limited to volume +/- support.

⁴ Only certain Apple devices can use the USB Host Mode lingo (Lingo 0x06). See ["Using an Apple Device as a USB Host"](#) (page 54).

Audio and Video Output Preferences

Various Apple devices support different settings and routings for video output and digital audio/video synchronization. To achieve the best results, an accessory must query the attached Apple device for its capabilities and set its audio and video output preferences as specified in this section. For information about synchronizing digital audio and video streams, see ["Audio/Video Synchronization"](#) (page 351).

Video Output Settings

The iAP General lingo includes commands that let automotive head units and other accessories determine if the Apple device supports video output, and if so, to control several video output preferences. All Apple devices let an accessory control the video screen configuration and format. Newer Apple devices also let the user control the video output connection type, aspect ratio, closed caption enabling, and subtitle behavior, as shown in [Table 1-8](#) (page 48).

Table 1-8 Video output capabilities

Capability	5G iPod	iPod classic; 3G, 4G, 5G, 6G nano	iPod touch, iPhone	2G, 3G touch, iPhone 3G, iPhone 3GS, iPad, iPad 3G, iPhone 4 (GSM model), 4G touch, iPhone 4 (CDMA model), iPad 2 (WiFi), iPad 2 (WiFi + 3G)
Widescreen signaling	No	Yes	No	
Composite (interlaced) output	Yes	Yes	Yes	
S-Video (interlaced) output	Yes	Yes	Yes	
Component (interlaced) output	No	No	Yes	No
Component (progressive) output	No	Yes	No	Yes
4:3 fullscreen aspect ratio	Yes	Yes	Yes	
16:9 widescreen aspect ratio	No	Yes	Yes	
Closed captioning	No	Yes	Yes	
Subtitle display	No	Yes	No	
Display resolution	640 x 480	NTSC: 720 x 480; PAL: 720 x 576		

An accessory can determine if the Apple device supports video output by sending a `GetiPodOptionsForLingo` command. If the Apple device does not support that command, the accessory may send a `GetiPodOptions` command. If the Apple device returns an `ACK` command with a Bad Parameter status, or sends a `RetiPodOptions` command with the Video Option bit (bit 0) clear, the Apple device does not support video. If the video option bit is set, the Apple device at least supports video output, screen configuration, and video format preferences. The accessory can determine if the Apple device supports other preferences by sending a `GetiPodPreferences` command for each preference. If the Apple device returns an `ACK` command with Bad Parameter status, that preference is not supported (see "[Command 0x29: GetiPodPreferences](#)" (page 120)).

Note: Accessories must use iAP commands to request analog audio output from an Apple device.

Character Encoding

The iTunes application and all Apple devices use Unicode UTF-8 encoding for tags and metadata. UTF-8 is compatible with the ASCII character set and supports other languages, including Chinese, Japanese, and Korean. It specifies unique encodings to represent all its supported character glyphs including the Roman alphabet and Han, Kanji, Kana, and Hangul characters. The UTF-8 format uses 1 to 4 bytes to represent each encoding; this means that a 5-character word can require from 5 to 20 bytes of storage. More information on Unicode and UTF-8 encoding can be found at <http://www.unicode.org/>.

Every accessory should decode UTF-8 characters. If the accessory is not capable of displaying certain Unicode character glyphs, a substitute glyph (such as an open square) should be displayed instead.

IMPORTANT: Accessories must make a special provision to decode and render the Unicode RIGHT SINGLE QUOTATION MARK character, which is used extensively by iTunes. This character appears as the 3-byte UTF-8 sequence 0xE2 0x80 0x99. Accessories must recognize and render this sequence using the RIGHT SINGLE QUOTATION MARK character (Unicode 0x2019) or convert it into the simple apostrophe character APOSTROPHE (Unicode 0x0027).

Accessory Control of iOS Devices

Accessory developers must take into account these current limitations on the ability of accessories to control an iOS device.

- An accessory cannot navigate the home screen of an iOS device.
- An accessory cannot unlock an iOS device's screen once it is locked.
- When the iOS device's screen is locked, most iAP commands continue to operate, but some Simple Remote lingo commands (such as the menu, select, and arrow button controls) produce no visible effect.
- Accessories are responsible for adjusting video out settings if they are not entering or using the Extended Interface mode. If it needs to display video outside the display on an iOS device, the accessory should set the video out preference to On, or to Ask if it wants the user to be prompted. In Remote UI mode, the Ask preference defaults to On and no prompt appears.

Accessory Launching of iOS Applications

A compatible accessory can request that an iOS device launch an application on its behalf. This action can improve the user's experience by automatically starting up the accessory's companion app when the accessory establishes communication with the iOS device.

For an accessory to use this autolaunch feature, all of the following conditions must be met:

- The iOS device must be running iOS 4 or later.
- The iOS device's Request Application Launch bit must be set; see General lingo bit 24 in [Table 2-138](#) (page 167). The accessory can verify this condition by sending a General lingo `GetiPodOptionsForLingo` command and checking the reply.
- The iOS device's display must be On.
- The iOS device must be unlocked.
- Either the iOS device's Home screen or the iOS app must be visible.

An accessory sends the General lingo command `RequestApplicationLaunch` (0x64) to launch an application on an iOS device. The accessory passes an Application ID string, such as "com.mycompany.myapp", to specify which application to launch. The iOS device replies with an ACK command that returns the application launch status. A status of OK (0x00) only indicates that the launch is possible; it does not guarantee that the application is running at that time. A status of Command Failed indicates that the application either does not exist on the iOS device or that the iOS device is in a condition that prevents the launch. The Accessory must not retry the `RequestApplicationLaunch` command if a Command Failed status is returned.

After its `RequestApplicationLaunch` is successfully acknowledged, the accessory must not assume that the requested app is actually running. Instead, the accessory must either wait to receive a General lingo `OpenDataSession` command for a protocol that it is expecting to use, or find that it is able to register for, and receive, a `NowPlayingAppFocusChange` notification from the app.

Note: The `NowPlayingAppFocusChange` notification is sent only by apps that explicitly support it. See Apple's iPhone Software Development Kit (SDK) for details about how to write apps with that support. If playing focus switches to an app that does not support the `NowPlayingAppFocusChange` notification, the accessory will not receive a notification.

The following General lingo commands support the accessory autolaunch feature:

- `RequestApplicationLaunch` (0x64) is sent by an accessory to an iOS device to request that it launch a specific application.
- `GetNowPlayingFocusApp` (0x65) is sent by an accessory to an iOS device to determine which application has the current Now Playing focus.
- `RetNowPlayingFocusApp` (0x66) is returned by the iOS device to the accessory to report which application has the current Now Playing focus.

Accessory Communication With iOS Applications

This section describes the communication process between accessories and iOS, and specifies the iAP commands that an accessory can use to open and maintain communication with an application.

Note: Before it can communicate with any application, an accessory must be identified through the IDPS process described in "Accessory Identification" (page 519) and must authenticate itself using Authentication 2.0B, as described in "Authentication" (page 71). Using IDPS also requires enabling transaction IDs in iAP commands, as described in "Transaction IDs" (page 525).

If the iOS device does not support IDPS, the accessory must send it an `IdentifyDeviceLingoes` command with all fields set to 0xFF. This causes the iOS device to display an "Accessory not supported" message.

The IDPS process lets an iOS device match its applications with its attached accessories, making it possible to open communication between them. Once an accessory has been identified and authenticated, the process described below can create a route for two-way data transmission between it and compatible applications.

The communication process between an accessory attached to an iOS device and an application running on the device requires both accessory iAP commands and application programming. The iAP side of the process is described in this section. The iOS side uses the External Accessory Framework, which is described in Apple's Software Development Kit for iOS.

Setting Up a Communication Session

To communicate with an application, an accessory must first expose certain protocol identifiers by sending `protocolString` values to the iOS device; see Table 2-85 (page 136). This information is used to establish communication channels to the application. The accessory must also send an Apple-assigned `BundleSeedIDString` value, as shown in Table 2-86 (page 137). This string identifies the accessory's preferred application. The Bundle Seed ID string is assigned to the application developer through the iPhone Developer Program, as part of the application development process. For further information, see "Managing Devices" in *iPhone Development Guide*, available at <http://developer.apple.com/iphone>. The accessory sends the `BundleSeedIDString` value as part of its startup identification, as described in "Accessory Identification" (page 519).

The design and maintenance of communication protocols between accessories and applications are entirely the responsibility of the developers of these products. Apple makes no attempt to secure protocol name space or provide for communication security between accessories and applications. Protocol names must be in reverse-DNS format and must be associated with domains that are registered with the Internet Corporation for Assigned Names and Numbers (ICANN), so that Apple can rely on each protocol having a unique owner. Apple does not require a developer to be the owner of the domain name used, but will require (by self-certification) that the developer has permission to use the domain name for the protocol. For suggestions on protocol design, see "Communication Protocol Design Hints" (page 52).

When an application requests access to an accessory, using the iOS External Accessory Framework, the iOS device sends it an `OpenDataSessionForProtocol` command. As a part of this notification, `sessionID` and `protocolIndex` values are sent to the accessory, so it can identify incoming iAP commands for this session. A session ID is included in all iAP data transfer commands, so that multiple sessions may run concurrently. The accessory should accept the connection with a `DevACK` response.

When the application has finished with the accessory, or the iOS device has determined that communication is no longer needed, a `CloseDataSession` command is sent to the accessory. The accessory must cease communication through the `sessionID` that was just closed. At this time the accessory can implement any power-saving features it may have.

Data Flow to and from the iOS Device

The iAP command `iPodDataTransfer` is used to communicate information to the accessory from the application and `DevDataTransfer` to communicate information to the application from the accessory. Both iAP commands contain a `sessionID` parameter to identify which channel they are traveling through. The message delivery order is first in, first out. End-to-end delivery of data is normally reliable, but not absolutely guaranteed, so developer protocols must be designed to recover from loss of data. For information about iAP transport links, see “Protocol Transport Links” in *MFi Accessory Hardware Specification*. For the rules that must be followed during a communication session, see “[Command 0x42: DevDataTransfer](#)” (page 148) and “[Command 0x43: iPodDataTransfer](#)” (page 150).

Matching Accessories With Applications

The iOS device matches applications and accessories by using the preferences and protocol names communicated by the accessory during its IDPS session. If the accessory names multiple protocols, it will be deemed compatible with an application if the application supports at least one of them. In addition, the accessory’s `BundleSeedIDString` value is used to identify preferred or default applications.

A `BundleSeedIDString` is required to properly pair the accessory to its preferred or default application on the iOS device. The application must be submitted to Apple as part of the accessory certification process. Apple assigns a Bundle seed ID value to the application developer, who provides this information to the accessory developer for inclusion in the accessory’s firmware.

Communication Protocol Design Hints

The owner or creator of a protocol for communication between accessories and applications must define the preferences that are required or optional. Each accessory and application must then negotiate their level of compatibility directly. Apple does not referee protocols, and protocol creators must ensure that their accessories and applications can verify that they support the same features. The following suggestions can help with protocol design:

- Ensure that the accessory can always resynchronize its data stream in the event of an error.
- The round-trip latency of commands may vary; do not draw inferences from propagation times. There is no guarantee of delivery timing over the communication link.
- After a connection has been established, clearly establish which end speaks first.
- Begin communication with an exchange of meta-information, such as confirmation of the protocol and version. Give both ends the opportunity to declare that they cannot proceed due to protocol incompatibilities.
- Ensure that the transaction order is clear and that both ends know their position in it. Packets within each stream are guaranteed to be ordered but not free of gaps between packets, so ensure that the stream can be resynchronized after a gap.
- Ensure that responses can always be matched to their corresponding requests. The simplest forms of reliable communication are call and response. Interleaved communication streams are more complicated and less desirable.
- Establish an unambiguous indicator of the start of each packet.
- Add security where needed, but recognize that it slows down and complicates each transaction.

- Ensure that the other end of each transaction has both received and understood the last packet.
- Ensure that each packet is received intact and error free by adding a CRC value, checksum, or the like.
- Ensure that incoming data can be cached until it can be processed.
- Try to make the protocol extensible for future needs.

Communication iAP Commands

The General lingo commands that support applications communicating with accessories are listed in [Table 1-9](#) (page 53) and documented in detail in "[Lingo 0x00: General Lingo](#)" (page 78). All commands are protocol version 1.09 and require authentication 2.0B.

Table 1-9 Accessory communication commands

CmdID	Name	Direction	Payload:bytes
0x3F	OpenDataSession-ForProtocol	Dev to Acc	{transID;2, sessionID;2, protocolIndex;1}
0x40	CloseDataSession	Dev to Acc	{transID;2, sessionID;2}
0x41	DevACK	Acc to Dev	{transID;2, ackStatus;1, cmdID;1}
0x42	DevDataTransfer	Acc to Dev	{transID;2, sessionID;2, data;<var>}
0x43	iPodDataTransfer	Dev to Acc	{transID;2, sessionID;2, data;<var>}
0x4A	iPodNotification	Dev to Acc	{transID;2, notificationType;1, waitTime;4, overflowTransID;2}

Note: The transID parameters in the foregoing table are described in "[Transaction IDs](#)" (page 525).

Controlling Applications Using the Simple Remote Lingo

An accessory connected to an iOS device can use the Simple Remote lingo `ContextButtonStatus` command to send button presses to an application. The application must register for Simple Remote commands, as explained in the iOS SDK. Applications can accept the following button press events from accessories:

- Play/Pause
- Pause
- Play/Resume
- Next Track
- Previous Track
- Begin Fast Forward
- Begin Rewind

Note: These button states are a subset of those listed in [Table 3-15](#) (page 199). Accessories that communicate with iOS devices must not use other button states.

For a consistent user experience, it is recommended that accessories limit themselves to sending only the button states listed above. However, an accessory may send a `SetEventNotification` command to receive `NowPlayingFocusApp` notifications; this can let it determine when the iOS application is active, thus making it possible to enable or disable other button states through iAP communication with the application.

Using an Apple Device as a USB Host

Certain Apple devices (see [Table 1-6](#) (page 45)) can operate in USB Host mode, an operating mode in which the Apple device acts as a USB bus host and the accessory acts as a USB device. The following hardware details of USB Host Mode are documented in the section “Using an Apple Device as a USB Host” of *MFi Accessory Hardware Specification*, Version R6 or later:

- Design requirements for accessories that use USB Host Mode.
- How the accessory puts the Apple device into and out of USB Host Mode.
- Handling USB digital audio streams between the accessory and the Apple device.
- Handling USB MIDI.

In USB Host Mode, an accessory can exchange iAP commands with the Apple device over the USB transport link. USB Host mode may be advantageous to an accessory by letting it use low-cost USB chipsets that can provide higher data transfer rates between the accessory and the Apple device.

Entering USB Host Mode on Startup

An accessory can be designed to use either hardware or firmware (but not both) to place an Apple device in USB Host Mode:

- Using hardware, the accessory can put the Apple device into USB Host mode as soon as it is connected. The required hardware configuration is described in *MFi Accessory Hardware Specification*. The accessory must immediately enter the Identify Device Preferences and Settings (IDPS) process and authenticate itself, using iAP commands over USB transport instead of UART.
- If the Apple device supports USB Host Mode lingo, the accessory can complete the IDPS and authentication processes and then put the Apple device into USB Host mode by sending the iAP commands described in “[Lingo 0x06: USB Host Mode Lingo](#)” (page 273), using only UART transport. [Table 1-10](#) (page 55) lists the iAP commands used to manage USB Host mode.

Note: Because the USB Host Mode lingo replaces the deprecated USB Host Control lingo with the same lingo ID and version number (see “[Command History of the USB Host Lingoes](#)” (page 274)), the accessory must not use lingo ID information to test for USB Host Mode support.

Table 1-10 USB Host mode commands

Cmd ID	Cmd name	Direction	Parameters
0x00	DevACK	Acc to Dev	{cmdStatus:1, cmdIDOrig:1}
0x04	NotifyUSBMode	Dev to Acc	{usbMode:1}
0x80	iPodACK	Dev to Acc	{cmdStatus:1, cmdIDOrig:1}
0x81	GetiPodUSBMode	Acc to Dev	none
0x82	RetiPodUSBMode	Dev to Acc	{usbMode:1}
0x83	SetiPodUSBMode	Acc to Dev	{usbMode:1}

Note: Once the Apple device is in USB Host mode it will remain in that mode regardless of whether the USB V_{BUS} supply is turned on or off.

Upgrading from UART Transport to USB Host Mode

An accessory may start up sending iAP commands over the UART transport link and upgrade to USB Host Mode if the Apple device supports it. This upgrade lets accessories support a wider range of Apple devices while still taking advantage of USB Host Mode whenever possible.

Accessories must meet all the following requirements to be able to upgrade from UART to USB Host Mode:

- The accessory must be designed to put the Apple device into USB Host Mode on startup by means of hardware, as specified in the section “Using an Apple Device as a USB Host” in *MFi Accessory Hardware Specification*.
- The accessory must not support the 3G iPod.
- The accessory must communicate on only one transport (UART or USB) at any given time.
- The accessory’s hardware must disconnect its USB D+/D- lines, or place them in a tristate condition, so they float when the USB transport is not active.

The accessory must use the process specified below to determine whether the Apple device supports USB Host Mode and to update it to that mode. It must execute this process every time it detects a transition on the Accessory Power pin of the 30-pin connector from low to high state.

1. Immediately upon detecting Accessory Power, the accessory must tristate (float) its USB D+/D- signals (not actively drive them high or low).
2. The accessory must begin its identification process over UART transport by sending a `StartIDPS` command. It must then wait for the Apple device to respond with a General lingo `ACK` command.

- If the Apple device responds with an ACK error status, IDPS and USB host mode are not supported; the accessory must use only the UART transport for iAP communications. The accessory must now exit this update process.
 - If the Apple device sends an ACK success status in response to the `StartIDPS` command, the IDPS process is supported. The accessory must proceed to Step 3.
3. The accessory must send a General lingo `GetiPodOptionsForLingo` command to the Apple device, passing a `LingoID` of 0x00 (General lingo). The accessory must then wait for the `RetiPodOptionsForLingo` response.
 - If the USB host mode support bit 27 (listed in [Table 2-138](#) (page 167)) returned by `RetiPodOptionsForLingo` is 0, the Apple device does not support USB host mode; the accessory must continue to communicate over the UART transport link. The accessory must now exit this update process.
 - If the USB host mode support bit 27 returned by `RetiPodOptionsForLingo` is 1, the Apple accessory supports hardware switching into USB Host Mode. The accessory must proceed to Step 4.
 4. The accessory must cancel the IDPS process that it began on the UART transport, by sending a General lingo `EndIDPS` command that passes an `accEndIDPSStatus` value of 3 (see [Table 2-104](#) (page 143)). After receiving an `IDPSStatus` response from the Apple device, the accessory must cease communication over the UART transport link.
 5. The accessory must enable its USB D+/D- lines at the 30-pin connector and set D+ active to indicate that it is a full-speed or high-speed USB device.
 6. The accessory must wait for the Apple device to enumerate the accessory as a USB full-speed or high-speed USB device, following the enumeration process specified in the USB 2.0 standard.

Packet Formats for iAP Commands

All iAP command packets transferred over the USB IN and OUT pipes must follow the formats specified in ["Command Packet Formats"](#) (page 76), except that the sync byte (byte 0) of each packet is unnecessary and should be omitted. If the accessory receives a value of 0xFF as the first byte of a packet, it must ignore that byte.

Data Transfers With an Apple Device

When the Apple device enters USB Host Mode, it detects and enable the accessory's USB iAP interface, which consists of an interrupt IN pipe, bulk IN pipe, and bulk OUT pipe. See *MFi Accessory Hardware Specification* for details.

While it is in USB Host mode, the Apple device continuously polls the USB interrupt IN pipe, waiting to receive a zero-length packet (ZLP) data transfer from the accessory. When it has iAP commands or data ready to be read on the bulk IN pipe, the accessory must signal the Apple device by sending a single ZLP data transfer on the interrupt IN pipe. The Apple device then reads the bulk IN pipe repeatedly, requesting the maximum bulk IN USB packet size, until the accessory returns less data than the maximum packet size.

Note: The sync byte (byte 0) of all iAP command packets transferred over USB is unnecessary and should be omitted. If the accessory receives a value of 0xFF for byte 0, it must ignore that byte.

The accessory does not need to send another ZLP data transfer on the interrupt IN pipe as long as it continues to return the maximum USB packet size in response to each bulk IN read. After the accessory returns less data than the maximum packet size, it must send another ZLP data transfer at the next interrupt IN poll interval if it has more data to be read on the bulk IN pipe. If the accessory does not have any bulk IN data to send to the Apple device, it should respond to the USB interrupt IN pipe read by sending a USB NAK packet.

The Apple device will continue to poll the interrupt IN pipe, even while a bulk IN read is currently in progress. If the accessory has no new data transfer to send after the current transfer completes, it should respond to interrupt IN pipe reads by sending USB NAK packets. If the accessory wants to begin a new data transfer after the current transfer completes, it must respond to the next interrupt IN pipe read with by sending a ZLP data transfer. When the Apple device receives a ZLP data transfer on its interrupt IN pipe while a bulk IN read is currently in progress, it will queue up a new bulk IN read internally, to be started immediately after the current bulk IN transfer finishes.

If the Apple device's internal buffer pool is full, it may temporarily stop reading the bulk IN pipe. This is normal behavior; the Apple device will resume reading the bulk IN pipe (as needed) after one or more internal buffers become available. If bulk IN reads stop, even though the maximum packet size was returned with the last read, the accessory must not send more than one additional ZLP data transfer to signal that its bulk IN pipe should be read.

In USB Host mode, the Apple device sends data to the accessory using the bulk OUT pipe. The accessory must return a USB ACK packet if the write operation is successful or return a USB NAK packet if it is not ready to accept data. If the accessory repeatedly returns a USB NAK packet for more than 1 second, the write operation will time out and information from the Apple device may be lost.

Note: The accessory must be able to handle bulk IN and bulk OUT data transfers that occur concurrently.

Sample Data Transfer to an Apple device

Table 1-11 (page 57) illustrates a typical USB data transfer sequence from the accessory to an Apple device in USB Host mode.

Table 1-11 Sample USB data transfer from accessory to host

Step	Apple device	Accessory
1	Poll the USB interrupt IN pipe.	
2		Return a USB NAK packet in response to the interrupt IN pipe read because the accessory has no data to send to the Apple device.
Continue steps 1 and 2 until the accessory has data to send to the Apple device. When the accessory has data to send, it goes to step 3 in response to the next interrupt IN pipe read.		
3		Return a ZLP data transfer on the interrupt IN pipe in response to the interrupt IN pipe read.

Step	Apple device	Accessory
4	Read the USB bulk IN pipe.	
5		Return a packet of the maximum USB packet size on the USB bulk IN pipe in response to the Apple device's bulk IN pipe read.
Continue steps 4 and 5 until the accessory will be returning a packet of less than the maximum USB packet size in response to the next bulk IN pipe read. When the accessory has less than the maximum USB packet size of data to return, it must go to step 6 in response to the next bulk IN pipe read.		
6		Return a packet of less than the maximum USB packet size (or a ZLP) on the USB bulk IN pipe. This tells the Apple device that the accessory is completing the current bulk IN pipe data transfer.
To initiate another bulk IN pipe data transfer immediately, the accessory must send a new ZLP transfer on the interrupt IN pipe (or have sent one while the current transfer was in process), effectively returning to step 3. Otherwise, it must return to step 1.		

Optimizing Data Transfers

To achieve maximum data transfer rates to the Apple device using iAP data transfers, accessory developers should observe these hints and cautions:

- After the accessory sends a ZLP data transfer to the Apple device on the interrupt IN pipe (to initiate a data transfer on the bulk IN pipe), the accessory should respond to each bulk IN read by returning a packet of the maximum USB packet size. The accessory should continue doing this until the last USB packet, when it may return a partial or empty packet.
- The accessory should avoid sending data payloads smaller than the maximum USB packet payload size, because afterward it must send another ZLP data transfer on the interrupt IN pipe to reinitiate the bulk IN read process. The latency time between a ZLP data transfer on the interrupt IN pipe and a bulk IN read can be as long as the polling interval, introducing additional data transfer delays and lowering throughput.
- If the bulk IN data packet being returned is exactly the size of the USB packet payload, the Apple device will go on to read the next USB packet on the bulk IN pipe. The accessory must respond to that read with a ZLP data transfer to ensure that the bulk IN pipe read process terminates properly.
- The iAP packets sent over USB may cross USB packet boundaries, and USB packets may cross iAP packet boundaries. If possible, multiple iAP packets should be concatenated or combined to fill each USB packet up to the maximum USB packet payload size, thereby minimizing the number of transitions between the interrupt IN and bulk IN pipes. Each of these transitions introduces an interrupt IN pipe polling interval latency, which should be avoided because it slows the overall data transfer throughput.
- When transferring data in iAP packets, accessories should send the largest iAP packets possible. This permits more data to be sent to the Apple device before a response is required (such as an iAP ACK command). Sending small packets and waiting for the Apple device to respond introduces unnecessary delays and lowers data throughput.

- Accessories must not send extra interrupt IN pipe ZLPs beyond those required to signal the Apple device to read the bulk IN pipe. If surplus ZLPs are sent, they may cause the Apple device to poll the bulk IN pipe unnecessarily, thereby reducing the USB bus bandwidth available for other purposes. When the accessory sends surplus ZLPs without available data to read on the bulk IN pipe, the Apple device applies a bulk IN read timeout of 1 second. The bulk IN read transfer from the accessory must finish before the timeout. If the transfer does not finish before the timeout, the accessory must send a new ZLP to restart the Apple device's read of the bulk IN pipe.

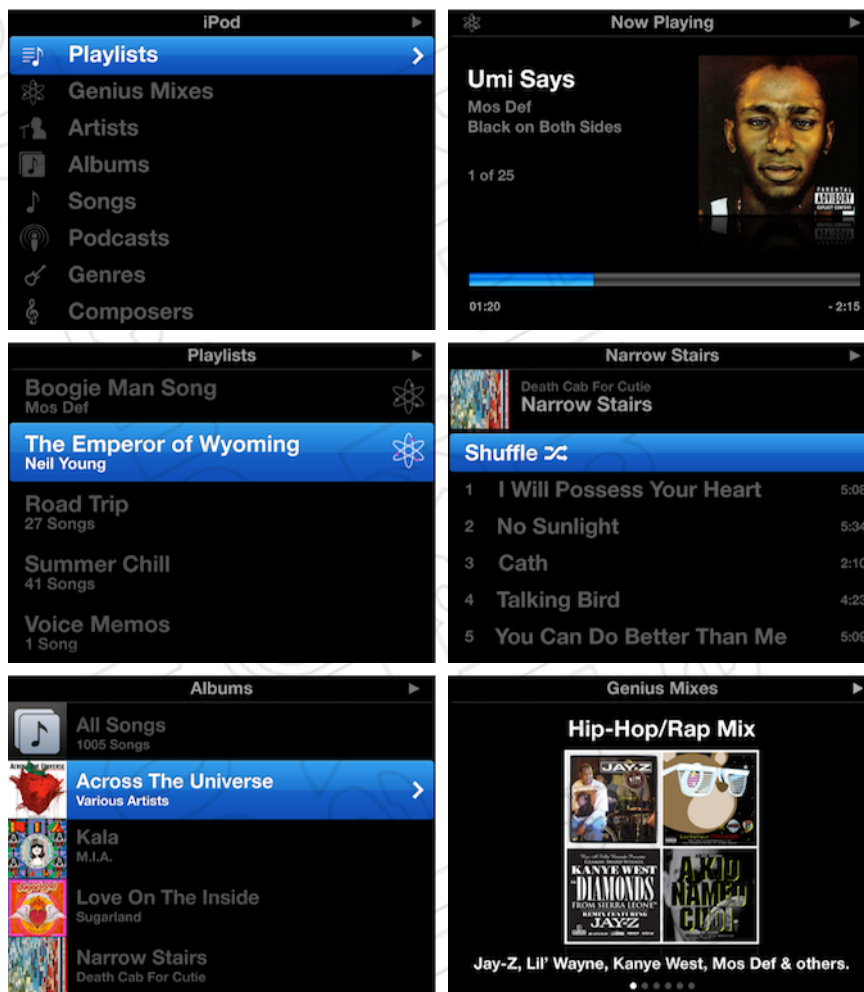
iPod Out Mode

iPod Out Mode is an Apple device operating mode that lets accessories interact with and control Apple devices through the native Apple user interface. While in this mode, the accessory supplies navigational inputs to the Apple device, using the Simple Remote or Display Remote lingo, and displays video output from the Apple device on a separate display. Additional support is provided by the iAP accessory lingo specified in "[Lingo 0x0D: iPod Out Lingo](#)" (page 374).

The primary advantage of iPod Out over other operating modes (such as Extended Interface mode) is that accessories can focus on passing user navigational inputs to the Apple device and displaying the Apple device's output video signal. Accessories that use iPod Out automatically take advantage of new user interface features in future Apple software, because the Apple device exports the new features to the accessory's display.

Screen shots of sample iPod Out displays are shown in [Figure 1-1](#) (page 60).

Figure 1-1 Sample iPod Out displays



Note: Accessory developers must observe the following requirements and cautions when using iPod Out Mode:

- Accessories must comply with the requirements of the section “iPod Out Accessory Requirements” in *MFi Accessory Hardware Specification*.
- For audio transport, the accessory may use either analog Line Out or digital audio over USB.
- The accessory must pass a `ScreenInfoToken` during its Identify Device Preferences and Settings (IDPS) process; see [Table 2-89](#) (page 138). It must also set its Video Out and analog audio transport preferences (if applicable) for iPod Out mode as part of the IDPS process.
- Some Simple Remote and Display Remote commands have no effect on an Apple device while it is in iPod Out mode.
- Currently, NTSC at 4:3 aspect ratio is the only video output format supported in iPod Out mode.
- Currently, 640 by 480 is the only video output resolution in iPod Out mode.

The Apple device exits iPod Out mode whenever one of the following events occurs:

- It receives a `SetUIMode` command that passes a `UIMode` value other than `0x02`.
- It receives an `EnterRemoteUIMode` command. This method should be used only if the Apple device does not support `SetUIMode`.
- It is detached from the accessory.
- The IDPS authentication process begins again while the Apple device is in iPod Out mode and the accessory's authentication state is reset.

To use iPod Out mode, an accessory must identify itself using IDPS and must authenticate itself using Authentication 2.0.

Setting iPod Out Video Preferences

To use iPod Out mode, an accessory must be attached to an Apple device that supports video output. The iAP General lingo contains commands that let accessories query Apple devices for their video capabilities and set Apple device video preferences. All Apple devices that have video output support control of the video screen configuration and format. Newer Apple devices also support selection of the video output connection type, aspect ratio, closed captioning, and subtitles.

An accessory can determine if an Apple device supports video output by sending it a `GetiPodOptions` command. If the Apple device replies with an `ACK` command that passes Bad Parameter status, or sends a `RetiPodOptions` command with the video option bit set to 0, it does not support video. If the video option bit is set to 1, the Apple device supports at least video output, screen configuration, and video format preferences. The accessory can determine if the Apple device supports additional video output preferences by sending a `GetiPodPreferences` command for each preference.

Before using iPod Out, an accessory must send the Apple device an IDPS `iPodPreferencesToken` value using the General lingo `SetFIDTokenValues` command. The class and setting IDs that must be passed in the token are listed in [Table 1-12](#) (page 61).

Table 1-12 Video preferences for iPod Out mode

Class ID	Preference	Setting	Setting ID	Description and comments
0x00	Video Out setting	To use iPod Out, the accessory must set this preference.		
		Reserved	0x00	
		On	0x01	The accessory must turn on Video Out before sending a <code>SetUIMode</code> command to enter iPod Out mode.
		Reserved	0x02-0xFF	
0x01	Reserved			
0x02	Video signal format	To use iPod Out, the accessory must set this preference.		
		NTSC	0x00	NTSC video format and timing.
		Reserved	0x01-0xFF	

Class ID	Preference	Setting	Setting ID	Description and comments
0x03-0x07	Reserved			
0x08	Video-out connection	To use iPod Out, the accessory must set this preference.		
		None	0x00	No connection.
		Composite	0x01	Composite video connection (interlaced video only).
		S-Video	0x02	S-Video video connection (interlaced video only).
		Component	0x03	Component Y/Pr/Pb video connection (interlaced or progressive scan, depending on the Apple device model).
		Reserved	0x04-0xFF	
0x09-0xFF	Reserved			

Supported Simple Remote and Display Remote Commands

While in iPod Out mode, Apple devices support a subset of the Simple Remote and Display Remote lingo commands. iPod Out accessories must not send any Simple Remote or Display Remote commands that do not appear in [Table 1-13](#) (page 62).

Table 1-13 Simple Remote and Display Remote commands supported in iPod Out mode

Lingo	Command	Comments
Simple Remote (Lingo 0x02)	iPodOutButtonStatus	
	RotationInputStatus	
	ContextButtonStatus	Accessories may send only these button states (see Table 3-15 (page 199)): Play/Pause Next Track Previous Track Play/Resume Pause Begin Fast Forward Begin Rewind
Display Remote (Lingo 0x03)	SetRemoteEventNotification	
	GetRemoteEventStatus	

Lingo	Command	Comments
	GetiPodStateInfo	
	GetPlayStatus	
	GetIndexedPlayingTrackInfo	
	GetNumPlayingTracks	

Improving User Interface Accessibility

Seven commands in the Simple Remote lingo (Lingo 0x02) help accessories improve an Apple device's user interface accessibility. They can allow users of the accessory to interact with the attached Apple device without needing to view or manipulate its touchscreen display. These commands are listed in "[User Interface Accessibility Commands](#)" (page 198).

The Protocol Core and the General Lingo

This chapter covers the basics of the iPod Accessory Protocol (iAP), including accessory identification and authentication. It also describes the packet format used for iAP commands and gives detailed descriptions of the commands included in the General lingo.

The iAP is used for both directions of a link, so every accessory must implement both sending and receiving capabilities. It should be possible to determine the direction (accessory to Apple device or Apple device to accessory) of a packet only from its contents. This means that no packet is valid for sending from both the Apple device and the accessory.

All accessories must be able to handle variable-length packets. An accessory must be able to accept a packet with extra data and ignore the unexpected bytes. At a minimum, the accessory must not lose sync to the packet signaling.

Most command packets generate a response, either an `ACK` command or a packet with data answering a query. Accessories should wait for the response before querying to see if the original request has been processed by the Apple device. In the Display Remote lingo, for example, when an accessory sends a `GetiPodStateInfo` command it should wait for the Apple device to return the corresponding `RetiPodStateInfo` (or `ACK`) command before sending a subsequent `GetPlayStatus` command.

Note: Unless otherwise stated in this specification, accessories must follow these timing rules:

- The accessory must wait at least 5000 ms for the Apple device to respond before retrying a command.
- When the Apple device sends a command for which it expects a response, the accessory must respond within 500 ms.
- If the accessory does not respond within 500 ms, it must not assume that the Apple device will retry the command.

Reserved Commands and Data

From time to time an Apple device may send an attached accessory a command that is reserved and not documented in this specification, or a documented command that has reserved data in its payload. Accessories must follow these rules when handling such reserved commands or data:

- When a data field is marked “Reserved” in this specification, accessories writing to it must set it to 0 (unless otherwise specified) and accessories reading it must ignore its value.
- If an accessory receives a command documented in this specification that passes reserved data, it must acknowledge successful receipt of the command, extract the data in it that is documented, and ignore the reserved data.

- If an accessory receives a command not documented in this specification, it should ignore it. Alternatively, it may return a General lingo ACK command with a value of 0x04 (Bad Parameter) to prevent the Apple device from waiting for a timeout period to receive a reply.

Accessory Signaling and Initialization

When attached, the accessory must detect the Apple device and initiate the identification process. This section describes the initialization process for both the UART serial port link and the USB port link.

Packet Signaling and Initialization Using the UART Serial Port Link

When using the UART serial port link, the identification process starts when the accessory detects current from the Apple device on the Accessory Power line of the 30-pin connector. The accessory must wait at least 80 ms and then transmit a sync byte. After sending the sync byte, the accessory must wait another 20 ms before starting the IDPS identification process.

IMPORTANT: To identify an accessory to an attached Apple device, before authentication, new accessory designs must use the Identify Device Preferences and Settings (IDPS) process described in Appendix B, "Accessory Identification" (page 519). This process ensures their compatibility with future firmware. Existing accessory designs may continue to send the iAP General lingo command 0x13, `IdentifyDeviceLingoes`, identifying the lingoes they support and requesting immediate authentication.

The IDPS accessory identification process begins when the accessory sends "Command 0x38: `StartIDPS`" (page 128) to the attached Apple device. If the Apple device does not respond, the accessory may resend `StartIDPS` command as long as Apple Device Detect (pin 30) is low and Accessory Power (pin 13) is high; see "Hardware Interfaces" in *MFi Accessory Hardware Specification*. The accessory must not retry IDPS identification until at least 500 ms has passed.

If the Apple device refuses the `StartIDPS` command by returning a General lingo ACK command with a status of 0x04 (Bad Parameter), the accessory must assume it is connected to an Apple device that doesn't support the IDPS process. In this case, the accessory must send an `IdentifyDeviceLingoes` command, requesting authentication, within 800 ms.

Table 2-1 (page 67) shows the command traffic for accessory identification and authentication when using the UART serial port link.

Note: Accessory identification and authentication may take place over any of the Apple device's transport links—UART, USB, or Bluetooth. If the process fails, the Apple device displays a "Accessory not supported" message to the user.

Table 2-1 Command traffic for UART accessory identification

Step	Action or command	Direction	Comments
1	Wait 80 ms after the Apple device turns on Accessory Power (pin 13 in "Hardware Interfaces" in <i>MFi Accessory Hardware Specification</i>)	Accessory	Wait for the Apple device's internal bootstrap and wakeup. If the Apple device is in Sleep mode, the accessory must repeat Steps 1-5 until the Apple device wakes and the accessory receives an ACK command.
2	Send sync byte (0xFF)	Acc to Dev	Allow the Apple device to synchronize to the accessory's baud rate.
3	Wait 20 ms	Accessory	
4	StartIDPS (0x38)	Acc to Dev	The accessory sends StartIDPS to start the identification process specified in "Accessory Identification" (page 519).
5	Wait up to 500 ms	Accessory	The accessory waits for the Apple device to send an ACK command acknowledging receipt of StartIDPS.
6	ACK (0x02) of StartIDPS	Dev to Acc	The Apple device sends a status of 0x00 (OK) to acknowledge receipt of the StartIDPS command.
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 7. Otherwise,</p> <ul style="list-style-type: none"> ■ If the accessory receives no ACK command within 500 ms, it may retry Step 1 at 500 ms intervals as long as Apple Device Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them. ■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the Apple device does not support IDPS. The accessory must not proceed to Step 7. Instead, it must perform either one of these two alternative actions: <ul style="list-style-type: none"> □ If IDPS support is required, within 800 ms the accessory must send an IdentifyDeviceLingoes command with all fields set to 0xFF. This causes the Apple device to display an "Accessory not supported" message. □ If IDPS support is not required, the accessory must then send another IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x00. See "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). A sample command sequence is listed in Table F-41 (page 621). ■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1. 			

Step	Action or command	Direction	Comments
7	SetFIDTokenValues (0x39)	Acc to Dev	The accessory sends the Apple device a collection of tokens that provide information about it; see " Command 0x39: SetFIDTokenValues " (page 130).
8	RetFIDTokenValueACKs (0x3A)	Dev to Acc	The Apple device acknowledges receipt of the tokens sent by the SetFIDTokenValues command.
9	EndIDPS (0x3B)	Acc to Dev	The accessory terminates the IDPS process, passing an accEndIDPSStatus value of 0x00 to ask the Apple device to continue with authentication; see " Command 0x3B: EndIDPS " (page 143).
10	IDPSStatus (0x3C)	Dev to Acc	The Apple device acknowledges receipt of EndIDPS and passes a status value of 0x00 to indicate that it has received all required tokens and that authentication will proceed; see " Command 0x3C: IDPSStatus " (page 144).
The Apple device now initiates the authentication process by sending a GetDevAuthenticationInfo command to the accessory, as shown in Table 2-5 (page 74).			

An accessory must reidentify itself if it receives a "[Command 0x00: RequestIdentify](#)" (page 83) command from the Apple device.

Once accessory packet transmission has begun, the maximum time between transmitted data bytes is 25 milliseconds. If the inter-character delay exceeds 25 ms, the Apple device discards any packet characters already received, plus any remaining characters received before the start of the next valid packet. The Apple device may exceed the 25 ms inter-character timing requirement in its outgoing packets when under heavy system load situations.

One known limitation exists when waking an Apple device from Sleep mode: the Apple device UART is not available for the first few milliseconds after waking from Sleep. If an accessory sends a packet to the Apple device while it is asleep, the first packet will be lost. Accessories must follow the steps below to wake an Apple device and ensure that the first packet is not lost:

1. Send a sync byte; this should wake the Apple device.
2. Wait for 20 ms.
3. Send the command packet with sync byte.

iAP Signaling and Initialization Using the USB or BT Port Link

When using iAP over USB, with the Apple device in USB Device Mode, initialization involves the USB host detecting the attached Apple device and setting its iUI configuration. Upon completion of this process, the host may send General lingo iAP commands to the Apple device using the USB HID interface. USB host access to the iAP accessory lingoes is enabled only after the host has identified itself and been authenticated by the Apple device. Before authentication, only a subset of the General lingo commands are available, as shown in [Table 2-4](#) (page 72).

For initialization with an iOS device in USB Host Mode, see [“Using an Apple Device as a USB Host”](#) (page 54).

To use iAP over Bluetooth, the accessory must first establish an RFCOMM protocol session with the Apple device. While the session is active, the host may send General lingo iAP commands to the Apple device using the RFCOMM channel. Access to the iAP accessory lingoes over BT is enabled only after the accessory has identified itself and been authenticated by the Apple device. Before authentication, only a subset of the General lingo commands are available, as shown in [Table 2-4](#) (page 72).

To identify an accessory to an Apple device, using USB or BT, new accessory designs must use the Identify Device Preferences and Settings (IDPS) process described in Appendix B, [“Accessory Identification”](#) (page 519). This process ensures their compatibility with future firmware. A sample command sequence is shown in [Table 2-2](#) (page 69). Existing accessory designs may continue to send the iAP General lingo command 0x13, `IdentifyDeviceLingoes`, identifying the lingoes they support and requesting immediate authentication.

Table 2-2 Commands for accessory identification via USB or BT

Step	Action or command	Direction	Comments
1	StartIDPS (0x38)	Acc to Dev	The accessory sends StartIDPS to start the identification process specified in “Accessory Identification” (page 519).
2	Wait up to 500 ms	Accessory	The accessory waits for the Apple device to send an ACK command acknowledging receipt of StartIDPS.
3	ACK (0x02) of StartIDPS	Dev to Acc	The Apple device sends a status of 0x00 (OK) to acknowledge receipt of the StartIDPS command.

Step	Action or command	Direction	Comments
<p>If the ACK reply to <code>StartIDPS</code> returns a status of 0x00, the accessory proceeds to Step 4. Otherwise,</p> <ul style="list-style-type: none"> ■ If the accessory receives no ACK command within 500 ms, it may retry Step 1 at 500 ms intervals as long as Apple Device Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them. ■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the Apple device does not support IDPS. The accessory must not proceed to Step 4. Instead, it must perform either one of these two alternative actions: <ul style="list-style-type: none"> □ If IDPS support is required, within 800 ms the accessory must send an <code>IdentifyDeviceLingoes</code> command with all fields set to 0xFF. This causes the Apple device to display an "Accessory not supported" message. □ If IDPS support is not required, the accessory must then send another <code>IdentifyDeviceLingoes</code> command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x00. See "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). A sample command sequence is listed in Table F-41 (page 621). ■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1. 			
4	<code>SetFIDTokenValues</code> (0x39)	Acc to Dev	The accessory sends the Apple device a collection of tokens that provide information about it; see " Command 0x39: SetFIDTokenValues " (page 130).
5	<code>RetFIDTokenValueACKs</code> (0x3A)	Dev to Acc	The Apple device acknowledges receipt of the tokens sent by the <code>SetFIDTokenValues</code> command.
6	<code>EndIDPS</code> (0x3B)	Acc to Dev	The accessory terminates the IDPS process, passing an <code>accEndIDPSStatus</code> value of 0x00 to ask the Apple device to continue with authentication; see " Command 0x3B: EndIDPS " (page 143).
7	<code>IDPSStatus</code> (0x3C)	Dev to Acc	The Apple device acknowledges receipt of <code>EndIDPS</code> and passes a status value of 0x00 to indicate that it has received all required tokens and that authentication will proceed; see " Command 0x3C: IDPSStatus " (page 144).
The Apple device now initiates the authentication process by sending a <code>GetDevAuthenticationInfo</code> command to the accessory, as shown in Table 2-5 (page 74).			

In addition to identifying itself at plug-in, a host may need to reidentify during an iAP session if the Apple device so requests. If the host receives "Command 0x00: RequestIdentify" (page 83), it must send a `StartIDPS` command and repeat the IDPS and authentication processes. USB hosts do not have to reconfigure iUI when responding to the `RequestIdentify` command from the Apple device.

Note: An accessory that is newly connected to a sleeping or hibernating Apple device cannot expect an immediate response. If it uses wired transport, the accessory should wait for the Apple device to turn on Accessory Power. If it uses Bluetooth or is unsuccessful the first time, it should retry the `StartIDPS` command every 500 ms until it gets a response.

Authentication

Authentication is a mechanism used by an Apple device to verify whether an attached accessory is an authorized accessory and by an accessory to authenticate the Apple device, if desired. Certain functionality on the Apple device is accessible only after an accessory has been authenticated as an authorized accessory. This functionality is summarized in Table 2-4 (page 72) and includes the use of any accessory lingo command over the USB or BT transport.

The limited set of General lingo commands listed in Table 2-4 (page 72) as requiring no authentication are free over USB and BT. After the accessory identification process, authentication may be independently initiated from either the Apple device or the accessory. In the identify and authentication sequences, the accessory should check the ACK status for all commands. If a failure status is returned, the process has failed and should be retried and/or abandoned.

Note: An accessory must not retry the IDPS process until at least 500 ms has passed.

IMPORTANT: To participate in the authentication process, an accessory must contain an authentication coprocessor provided by Apple.

Levels of Accessory Authentication

iAP supports three levels of security in the process by which an Apple device may authenticate an accessory connected to it:

- **None.** Older Apple device models through the iPod mini do not support authentication. They operate freely with simple accessories that do not require authentication, but they cannot work with newer or more sophisticated accessories whose range of functionality requires authentication. For details, see "Functional Description" in *MFi Accessory Hardware Specification*.
- **Authentication 1.0.** Version 1.0 authentication is based on public and private keys, where each accessory has a private key and every Apple device has the associated public key. The accessory authentication process is a part of the iAP General lingo (0x00) command protocol and is controlled by the Apple device's internal software. Accessories identify themselves to the Apple device as speaking specific lingo and supporting authentication. iAP queries the accessory's authentication information, sends a challenge to the accessory, and verifies that the accessory responds to the challenge correctly.

- **Authentication 2.0.** Version 2.0 authentication is based on standard X.509 Version 3 certificates. Information about this standard can be found at the IETF website: <http://tools.ietf.org/html/rfc3280>. Each certificate is generated and signed by a recognized certificate authority (CA) and has a unique serial number. Authentication 2.0 uses certificate classes to identify the type of accessory being authenticated, as shown in Table 2-3 (page 72).

IMPORTANT: Current Apple devices support Authentication 1.0, as shown in "Authentication 1.0 Sample Command Sequence" (page 619), only as a legacy technology, to make them compatible with existing accessories. All new accessory designs must use Authentication 2.0.

Table 2-3 Apple device authentication coprocessor classes

iPod Class ID	iPhone Class ID	Description
1	4	Automotive use only; enables all authenticated features.
2	5	Nonautomotive use only; enables all authenticated features except the Digital Audio lingo. Deprecated; do not use in new products.
3	6	Nonautomotive use only; enables all authenticated features.

Authentication Requirements

Because accessory authentication can take a significant amount of time (up to 75 seconds for Authentication 2.0A), it is performed as a background process. The process starts when the Apple device sends a `GetDevAuthenticationInfo` command to the accessory. The accessory must respond by transmitting its entire `RetDevAuthenticationInfo` command within 1.00 seconds for Authentication 1.0 or 2.00 seconds for Authentication 2.0, including the transmission of all sections of its certificate. When the Apple device transmits a `GetDevAuthenticationSignature` command, the accessory must transmit its `RetDevAuthenticationSignature` response within 7.00 seconds for Authentication 1.0, 75.00 seconds for Authentication 2.0A, or 2.00 seconds for Authentication 2.0B.

All lingo-authenticated commands and features are available to accessories once the Apple device has sent the accessory an `AckDevAuthenticationInfo` command with success status (0x00). This provisional authentication lasts until the Apple device sends a `AckDevAuthenticationStatus` command indicating that authentication has finished. The lingo command availability will be revoked if the authentication process fails, the accessory reidentifies without authentication, the Apple device sleeps or powers on, or the accessory is detached from the Apple device. The Apple device starts both a timer and a retry counter to guarantee that the authentication process will conclude.

Table 2-4 Lingo commands requiring authentication

Lingoes	UART transport	USB transport	BT transport
Lingo 0x00: General	No (0x00-0x10, 0x13-0x19, 0x23-0x25, 0x27-0x28, 0x38-0x3C, 0x46-0x48) Yes (0x11-0x12, ¹ 0x1A-0x1F, 0x29-0x2B, 0x35-0x37, 0x3F-0x43, 0x49-0x51, ¹ 0x54, 0x64-0x66 ²)		
Lingo 0x01: Microphone	Deprecated		

Lingoes	UART transport	USB transport	BT transport
Lingo 0x02: Simple Remote	No (0x00) Yes (0x01, 0x03–0x04, 0x0B–0x1A, 0x81)	Yes	Yes ³
Lingo 0x03: Display Remote	No (0x00–0x07, 0x1A–0x1E) Yes (0x08–0x19, 0x1F–0x22)	Yes	Yes ³
Lingo 0x04: Extended Interface	No (0x00–0x3A) Yes (0x3B–0x45, 0x47–0x49)	Yes	Yes ³
Lingo 0x05: Accessory Power	No	Yes	N/A
Lingo 0x06: USB Host Mode	Yes	N/A	N/A
Lingo 0x07: RF Tuner	Yes	Yes	N/A
Lingo 0x08: Accessory Equalizer	Yes	Yes	N/A
Lingo 0x09: Sports	Yes	Yes	Yes
Lingo 0x0A: Digital Audio	Yes	Yes	N/A
Lingo 0x0B: Reserved	N/A	N/A	N/A
Lingo 0x0C: Storage	Yes	Yes	Yes
Lingo 0x0D: iPod Out	Yes	Yes	Yes
Lingo 0x0E: Location	Yes	Yes	Yes
Lingoes 0x0F–0x1F: Reserved	N/A	N/A	N/A

¹ Apple devices will accept the General lingo commands 0x11 (RequestTransportMaxPayloadSize) and 0x4B (GetiPodOptionsForLingo) from an accessory during the IDPS process, before authentication. After the IDPS process these commands must be authenticated.

² Preference commands (0x29–0x4C) require authentication on all Apple devices except the 5G iPod; however, getting or setting the line-out preference class (0x03) does not require authentication. Commands 0x42 (DevDataTransfer) and 0x43 (iPodDataTransfer) are not available until authentication has finished.

³ Bluetooth accessories that support the Simple Remote, Display Remote, or Extended Interface lingoes must have their own volume controls and not use iAP volume control commands. They must also implement Extended Interface playback status notifications, as specified in "Playback Status Notifications" (page 418).

Apple Device Authentication of the Accessory

The process of authenticating the accessory is initiated by the Apple device, based on the settings made in the IDPS process. The authentication options also allow an accessory to request authentication of an Apple device, as described in "Accessory Authentication of the Apple Device" (page 75).

IMPORTANT: The accessory must send the Apple device a `StartIDPS` command before it tries to send any other iAP commands.

Table 2-5 (page 74) summarizes the authentication process, using Authentication 2.0. This is the authentication process that all new accessories should use. Steps 4 and 5 are necessary only if the accessory is unable to use the IDPS process and is initializing using `IdentifyDeviceLingoes`.

Table 2-5 Command traffic for accessory authentication

Step	Action or command	Direction	Comments
1	<code>GetDevAuthentication-Info (0x14)</code>	Dev to Acc	The Apple device requests accessory authentication information and starts its timeout timer.
2	<code>RetDevAuthentication-Info (0x15)</code>	Acc to Dev	The accessory returns its major and minor authentication version and its X.509 public certificate (see Note 1, below).
3	<code>AckDevAuthentication-Info (0x16)</code>	Dev to Acc	The Apple device assembles and checks the accessory's X.509 certificate. If it does not support the authentication version, or if the certificate check fails, the Apple device sends a value of 0x08 to the accessory (see Note 2, below).
All lingo-authenticated commands are enabled after the authentication version is validated, the lingoes requested by the accessory are checked against the lingoes allowed by the X.509 certificate, and the certificate has been verified (see Note 3, below).			
4 (non-IDPS only)	<code>GetAccessoryInfo (0x27)</code>	Dev to Acc	The Apple device queries the accessory for information about it.
5 (non-IDPS only)	<code>RetAccessoryInfo (0x28)</code>	Acc to Dev	The accessory returns information about its identity and capabilities.
6	<code>GetDevAuthentication-Signature (0x17)</code>	Dev to Acc	The Apple device sends a 20-byte random challenge to the accessory and asks it to calculate a digital signature.
7	<code>RetDevAuthentication-Signature (0x18)</code>	Acc to Dev	The accessory returns its digital signature to the Apple device within 2 seconds (Authentication 2.0B) or 75 seconds (Authentication 2.0A).
8	<code>AckDevAuthentication-Status (0x19)</code>	Dev to Acc	The Apple device verifies the signature, using the public key contained in the accessory's X.509 certificate, and returns the status of signature verification.

Notes:

1. Not more than 500 certificate bytes may be sent at once. If the X.509 certificate is larger than 500 bytes, the accessory must divide it into sections, each not larger than 500 bytes, for reassembly by the Apple device. The Apple device will send a General lingo ACK command for each certificate section up to, but not including, the last one. The accessory must wait for the ACK command before sending the next certificate section in a `RetDevAuthenticationInfo` packet.
2. The ACK response to each certificate section only acknowledges receipt. When all the sections have been assembled, the Apple device evaluates the whole certificate and sends an `AckDevAuthenticationInfo` command that states whether or not it is valid.
3. The Apple device parses the X.509 certificate into an allowed lingoes mask. It uses the mask to confirm that the accessory's identified lingoes are allowed by the certificate. If the accessory requests lingoes not allowed by the certificate, authentication fails. The Apple device also verifies that the certificate is valid.
4. If authentication fails, the accessory should repeat the whole IDPS and authentication process by sending `StartIDPS` again, as described in ["Accessory Signaling and Initialization"](#) (page 66).

The Apple device and the accessory can perform noncritical operations while background authentication is in progress. The command timeout counter and retry counter are not reset until authentication is complete or has failed.

Some lingoes requested by an accessory (such as the General, Accessory Equalizer, and Sports lingoes) cause the Apple device to request accessory information after it sends the `AckDevAuthenticationInfo` command. Accessories should be able to handle both authentication requests and asynchronous information requests from the Apple device during the authentication process.

With IDPS all Apple device preferences are set before authentication. Accessories that must use `IdentifyDeviceLingoes` should set any Apple device preferences that do not require authentication within 2 seconds of receiving the ACK reply to `IdentifyDeviceLingoes`, and set Apple device preferences that require authentication within 2 seconds of receiving the `AckDevAuthenticationInfo` command.

IMPORTANT: If the accessory is disconnected during the authentication process, or if authentication does not finish for other reasons, the Apple device may refuse to recognize the accessory for several minutes. With the IDPS process, the first command issued should be `StartIDPS`. This will clear the Apple device's state and allow normal identification and authentication to occur. If the Apple device does not support IDPS and the accessory's mode of use is such that it might be disconnected and then reconnected during the authentication process, then the accessory should send the Apple device an `IdentifyDeviceLingoes` command with the No Authentication option (see [Table 2-34](#) (page 98)) as its first command every time it is connected. This would cancel any authentication process that might already be running in the Apple device from a previous accessory attachment.

If the Apple device fails to authenticate the accessory at any point in the Authentication 1.0 or 2.0 process, and the retry count is exhausted, the Apple device may display an "Accessory not supported" message to the user.

Accessory Authentication of the Apple Device

The accessory can initiate a process to authenticate the Apple device, any time after it has been authenticated by the Apple device. This process is summarized in [Table 2-6](#) (page 76). See ["General Lingo Command Summary"](#) (page 79) for information on the individual commands.

Note: Accessory authentication of the Apple device is currently supported only for Authentication 2.0.

Table 2-6 Accessory authentication of the Apple Device

Step	Action or command	Direction	Comments
1	GetiPodAuthentication-Info (0x1A)	Acc to Dev	The accessory requests information to authenticate the Apple device.
2	RetiPodAuthentication-Info (0x1B)	Dev to Acc	The Apple device returns its major and minor authentication version and its X.509 public certificate (see Note, below).
3	AckiPodAuthentication-Info (0x1C)	Acc to Dev	The accessory assembles and checks the Apple device's X.509 certificate. If the accessory does not support the authentication version, or if the certificate check fails, it returns an error status.
4	GetiPodAuthentication-Signature (0x1D)	Acc to Dev	The accessory sends a 20-byte random challenge to the Apple device and asks it to calculate a digital signature.
5	RetiPodAuthentication-Signature (0x1E)	Dev to Acc	The Apple device returns its digital signature to the accessory within 75 seconds. The accessory verifies the signature, using the public key contained in the Apple device's X.509 certificate. If verification succeeds, the accessory begins to operate.
6	AckiPodAuthentication-Status (0x1F)	Acc to Dev	The accessory returns the status of the digital signature comparison.

Note:

1. The Apple device's X.509 certificate is sent in a PKCS-7 message containing only the certificate, encoded in DER format. See the IETF document RFC 2311, "S/MIME Version 2 Message Specification" for details about using PKCS-7 messages to transfer X.509 certificates. If necessary, the Apple device divides the X.509 certificate into sections, each not larger than the maximum payload size that the accessory specified during the IDPS process, for reassembly by the accessory. If the accessory has not specified a payload size, the Apple device sends 128-byte sections. The accessory must not acknowledge any certificate sections sent by the Apple device until the complete certificate has been received and verified.

Command Packet Formats

This section describes the general format for iAP packets. Use the small packet format for payloads up to 255 bytes; use the large packet format for payloads greater than 255 bytes. To determine the maximum payload size per packet allowed by the current transport (UART, USB, or USB Host mode), send "[Command 0x11: RequestTransportMaxPayloadSize](#)" (page 95).

Small Packet Format

For command packets whose payloads are 255 bytes or less, use the small packet format. The small packet format is shown in [Table 2-7](#) (page 77).

Table 2-7 Small packet format

Byte number	Value	Meaning
0x00	0xFF	Sync byte (required only for UART transport)
0x01	0x55	Packet start byte
0x02	0xNN	Packet payload length
0x03	0xNN	Lingo ID
0x04	0xNN	Command ID
0x05...0xNN	0xNN	Command data
(last byte)	0xNN	Packet payload checksum

Note that the command ID and command data format for packets with currently unspecified lingoes may not follow the format indicated here (1 byte command ID, 0xN bytes command data). Also note that a packet payload length of 0x00 is not valid for the small packet format; it is reserved as a marker for the large packet format.

Large Packet Format

For command packets whose payloads are between 256 bytes and 65535 bytes in length, use the large packet format. The large packet format is shown in [Table 2-8](#) (page 77).

Table 2-8 Large packet format

Byte number	Value	Meaning
0x00	0xFF	Sync byte (required only for UART transport)
0x01	0x55	Packet start byte
0x02	0x00	Packet payload length marker
0x03	0xNN	Packet payload length (bits 15:8)
0x04	0xNN	Packet payload length (bits 7:0)
0x05	0xNN	Lingo ID
0x06	0xNN	Command ID
0x07...0xNN	0xNN	Command data

Byte number	Value	Meaning
(last byte)	0xNN	Packet payload checksum

Packet Details

The sync byte (0xFF) is not considered part of the packet. It is sent merely to facilitate automatic baud rate detection and correction when using a UART serial port link and, in some cases, to power-on the Apple device. It is not necessary to send the sync byte when using BT or USB as a link.

The packet payload length is the number of bytes in the packet, not including the sync byte, packet start byte, packet payload length byte, or packet payload checksum byte. That is, it is the length of the command ID, lingo, transaction ID bytes (if any), and command data. Thus, the packet payload data length for a `RequestIdentify` command would be 0x02 without a transaction ID or 0x04 with. The Lingo ID specifies the broad category that the communication falls under. The Command ID is a more specific indication of the significance of the packet and is interpreted differently depending on the Lingo ID.

Unless otherwise specified, the following rules apply:

- All packet data fields larger than 8 bits are sent and received in big-endian format; that is, ordered from the most significant byte to the least significant byte.
- Accessory command packets that have a valid checksum but contain an invalid parameter, invalid command, or other such failure cause the Apple device to respond with an ACK command containing the appropriate error status.
- A packet with an invalid checksum received by an Apple device is presumed to be invalid and is ignored. No ACK or other command is sent to the accessory in response to the invalid packet.

Note: Unless otherwise specified, all data units larger than bytes must be transferred in a big-endian order; that is, 32 bits should be sent as bits 31:24, followed by bits 23:16, and so forth. Similarly, 16 bits should be sent as bits 15:8, followed by bits 7:0. This includes the 16-bit large packet format payload length: the high byte of the length is sent first, followed by the low byte of the length.

The sum of all the bytes from the packet payload length (or marker, if applicable) to and including the packet payload checksum is 0x00. The checksum must be calculated appropriately, by adding the bytes together as signed 8-bit values, discarding any signed 8-bit overflow, and then negating the sum to create the signed 8-bit checksum byte. All packets received with a nonzero checksum are presumed to be corrupted and will be discarded.

Lingo 0x00: General Lingo

The General lingo is intended for housekeeping commands and must be supported by all accessories.

General Lingo Command Summary

Table 2-9 (page 79) gives a summary of all commands in the General lingo, including the command ID, the length of the associated data, and the first version of the General lingo protocol in which the command was supported.

Table 2-9 General lingo commands

Command	ID	Data length without transaction ID	First protocol version
RequestIdentify	0x00	0x00	All
Identify (deprecated ; see "General Lingo Command 0x01: Identify" (page 598))	0x01	0x01	All
ACK	0x02	0x02 or 0x06	1.00
RequestRemoteUIMode	0x03	0x00	1.00
ReturnRemoteUIMode	0x04	0x01	1.00
EnterRemoteUIMode	0x05	0x00	1.00
ExitRemoteUIMode	0x06	0x00	1.00
RequestiPodName	0x07	0x00	1.00
ReturniPodName	0x08	0xNN	1.00
RequestiPodSoftwareVersion	0x09	0x00	1.00
ReturniPodSoftwareVersion	0x0A	0x03	1.00
RequestiPodSerialNum	0x0B	0x00	1.00
ReturniPodSerialNum	0x0C	0xNN	1.00
RequestiPodModelNum (deprecated ; use GetiPodOptionsForLingo)	0x0D		
ReturniPodModelNum (deprecated ; use RetiPodOptionsForLingo)	0x0E		
RequestLingoProtocolVersion	0x0F	0x01	1.00
ReturnLingoProtocolVersion	0x10	0x03	1.00
RequestTransportMaxPayloadSize	0x11	0x00	1.09
ReturnTransportMaxPayloadSize	0x12	0x02	1.09
IdentifyDeviceLingoes	0x13	0x0C	1.01
GetDevAuthenticationInfo	0x14	0x00	1.01

Command	ID	Data length without transaction ID	First protocol version
RetDevAuthenticationInfo	0x15	0xNN	1.01
AckDevAuthenticationInfo	0x16	0x01	1.01
GetDevAuthenticationSignature	0x17	0x11 or 0x15	1.01
RetDevAuthenticationSignature	0x18	0xNN	1.01
AckDevAuthenticationStatus	0x19	0x01	1.01
GetiPodAuthenticationInfo	0x1A	0x00	1.01
RetiPodAuthenticationInfo	0x1B	0xNN	1.01
AckiPodAuthenticationInfo	0x1C	0x01	1.01
GetiPodAuthenticationSignature	0x1D	0xNN	1.01
RetiPodAuthenticationSignature	0x1E	0xNN	1.01
AckiPodAuthenticationStatus	0x1F	0x01	1.01
Reserved	0x20–0x22	N/A	N/A
NotifyiPodStateChange	0x23	0x01	1.02
GetiPodOptions	0x24	0x00	1.05
RetiPodOptions	0x25	0x08	1.05
Reserved	0x26	N/A	N/A
GetAccessoryInfo	0x27	0xNN	1.04
RetAccessoryInfo	0x28	0xNN	1.04
GetiPodPreferences	0x29	0x01	1.05
RetiPodPreferences	0x2A	0x02	1.05
SetiPodPreferences	0x2B	0x03	1.05
Reserved	0x2C–0x34	N/A	N/A
GetUIMode	0x35	0x00	1.09
RetUIMode	0x36	0x01	1.09
SetUIMode	0x37	0x01	1.09
StartIDPS	0x38	0x00	1.09
SetFIDTokenValues	0x39	0xNN	1.09

Command	ID	Data length without transaction ID	First protocol version
RetFIDTokenValueACKs	0x3A	0xNN	1.09
EndIDPS	0x3B	0x01	1.09
IDPSStatus	0x3C	0x01	1.09
Reserved	0x3D–0x3E	N/A	N/A
OpenDataSessionForProtocol	0x3F	0x05	1.09
CloseDataSession	0x40	0x04	1.09
DevACK	0x41	0x04	1.09
DevDataTransfer	0x42	0xNN	1.09
iPodDataTransfer	0x43	0xNN	1.09
Reserved	0x44–0x45	N/A	N/A
SetAccStatusNotification	0x46	0x04	1.09
RetAccStatusNotification	0x47	0x04	1.09
AccessoryStatusNotification	0x48	0xNN	1.09
SetEventNotification	0x49	0x0A	1.09
iPodNotification	0x4A	0xNN	1.09
GetiPodOptionsForLingo	0x4B	0x01	1.09
RetiPodOptionsForLingo	0x4C	0x09	1.09
GetEventNotification	0x4D	0x00	1.09
RetEventNotification	0x4E	0x08	1.09
GetSupportedEventNotification	0x4F	0x00	1.09
CancelCommand	0x50	0x05	1.09
RetSupportedEventNotification	0x51	0x08	1.09
Reserved	0x52–0x53	N/A	N/A
SetAvailableCurrent	0x54	0x02	1.09
Reserved	0x55–0x63	N/A	N/A
RequestApplicationLaunch	0x64	0xNN	1.09
GetNowPlayingFocusApp	0x65	0x00	1.09

Command	ID	Data length without transaction ID	First protocol version
RetNowPlayingFocusApp	0x66	0xNN	1.09
Reserved	0x67–0xFF	N/A	N/A

When first connected to an Apple device, every accessory must identify itself. New accessory designs must use the Identify Device Preferences and Settings (IDPS) process described in Appendix B, "[Accessory Identification](#)" (page 519). This process ensures their compatibility with future firmware. Existing accessory designs may send the iAP General lingo command 0x13, `IdentifyDeviceLingoes`, identifying the lingoes they support. For details, see "[Accessory Signaling and Initialization](#)" (page 66).

The Apple device may send a `RequestIdentify` command to the accessory to ask it to reidentify itself. There is currently no data defined for this command. Any command returned in response to a `RequestIdentify` packet does not need to have the extra sync bytes and delays used during the accessory startup process (described in "[Accessory Signaling and Initialization](#)" (page 66)).

The remaining General lingo commands can be used to obtain general information from the Apple device. These commands allow the accessory to request the name, serial number, model number, and software version number of the Apple device. The `RequestLingoProtocolVersion` command allows an accessory to query the Apple device for the lingo protocol versions of all supported lingoes on the Apple device. The `ACK` command is used by the Apple device to report command error conditions and has an ACK pending feature to notify the requesting accessory how long to wait for responses to certain commands.

Accessories should send the `GetiPodOptionsForLingo` command to query the Apple device for its support of specific features for each lingo. The Apple device responds with a `RetiPodOptionsForLingo` command, telling the accessory exactly which options it supports for the specific lingo. With Apple devices that do not support `GetiPodOptionsForLingo`, an accessory can send `RequestLingoProtocolVersion` to get the lingo version; however, this method forces the accessory to infer features using the method described in "[Protocol Features and Availability](#)" (page 41).

History of the General lingo protocol

[Table 2-10](#) (page 82) lists changes introduced with each version of the General lingo protocol.

Table 2-10 General lingo revision history

Lingo version	Command changes	Features
No version	Add: 0x00, 0x01	Request identification, identify as single lingo accessory
1.00	Add: 0x02, 0x07–0x10	ACK response, request Apple device name, software version, serial number, and model number; support for 38400 and 57600 baud rates
1.01	Add: 0x13–0x1F	Identify as multilingo accessory, Authentication 1.0 process
1.02	Add: 0x23	Notify accessory of Apple device state changes (Sleep/Power On/Hibernate)
1.03	None	(Internal code restructuring)

Lingo version	Command changes	Features
1.04	Add: 0x27, 0x28	Get/return accessory information, Authentication 2.0 process
1.05	Add: 0x24–0x25, 0x29–0x2B	Video browsing preferences
1.06	None	Line-out preferences, two-way Authentication 2.0
1.07	None	Additional video preferences
1.08	None	Different timeout times for <code>GetDevAuthenticationInfo</code> and <code>GetDevAuthenticationSignature</code> .
1.09 (see Note, below)	Add: 0x11–0x12, 0x35–0x3C, 0x3F–0x43, 0x4A–0x4C, 0x50, 0x64–0x66	Report maximum payload sizes, Identify accessories through IDPS, communicate with applications, query Apple device for lingo options, support USB Host mode, support iPod Out, support accessory autolaunch.

Note: The accessory must use `GetiPodOptionsForLingo` to determine whether the attached Apple device supports specific features of lingo version 1.09.

Command 0x00: RequestIdentify

Direction: Apple device to Accessory

The Apple device sends this command to prompt accessories to reidentify themselves. If an accessory receives this command, it must respond with `StartIDPS`, as described in "[Accessory Signaling and Initialization](#)" (page 66).

Table 2-11 RequestIdentify packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo. All accessories must support this lingo.
4	0x00	Command: RequestIdentify
5	0xFE	Checksum

Command 0x02: ACK

Direction: Apple device to Accessory

The Apple device sends the ACK command to notify the accessory of command completion status and errors. The ACK command may come in one of several forms, depending on the status being returned. For any status other than command pending or dropped data, the ACK packet shown in [Table 2-12](#) (page 84) is used. If transaction IDs are enabled, the ACK packet shown in [Table 2-13](#) (page 84) is used. Other formats are used for command pending and dropped data, as shown in [Table 2-14](#) (page 85). Transaction IDs are described in ["Using Transaction IDs"](#) (page 525).

Table 2-12 ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command: ACK
5	0xNN	Command result status. Possible values are shown in Table 2-14 (page 85).
6	0xNN	The ID of the command being acknowledged
7	0xNN	Checksum

Table 2-13 ACK packet with transaction IDs

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command ID: ACK
5	0xNN	transID [bits 15:8]: Transaction ID; see "Using Transaction IDs" (page 525)
6	0xNN	transID [bits 7:0]
7	0xNN	ackStatus: Status of command received. See Table 2-14 (page 85).
8	0xNN	origCmdID: ID of command received.
9	0xNN	Checksum

Table 2-14 ACK command error codes

Value	Description
0x00	Success (OK)
0x01	ERROR: Unknown database category
0x02	ERROR: Command failed
0x03	ERROR: Apple device out of resources
0x04	ERROR: Bad parameter
0x05	ERROR: Unknown ID
0x06	Command Pending; see Table 2-15 (page 86) and Table 2-16 (page 86)
0x07	ERROR: Not authenticated
0x08	ERROR: Bad authentication version
0x09	ERROR: Accessory power mode request failed
0x0A	ERROR: Certificate invalid
0x0B	ERROR: Certificate permissions invalid
0x0C	ERROR: File is in use
0x0D	ERROR: Invalid file handle
0x0E	ERROR: Directory not empty
0x0F	ERROR: Operation timed out
0x10	ERROR: Command unavailable in this Apple device mode
0x11	ERROR: Accessory Detect not grounded or invalid Accessory Identify Resistor detected
0x12	ERROR: Accessory not grounded
0x13	Multisection data section received successfully; see "Multisection Data Transfers" (page 531)
0x14	ERROR: Lingo busy
0x15	ERROR: Maximum number of accessory connections already reached
0x16	Reserved
0x17	ERROR: Dropped data (SessionWriteFailure); see Table 2-17 (page 87)
0x18	ERROR: Attempt to enter iPod Out mode with incompatible video settings
0x19–0xFF	Reserved

If the status returned by the ACK command is Command Pending, an additional field is added to the ACK packet that represents the amount of time, in milliseconds, that an accessory should wait to receive the final packet indicating that the current command completed or returned an error status.

After receiving a Command Pending ACK, the accessory should wait for up to the specified number of milliseconds for a final ACK response. If no final ACK packet is received before the specified amount of time expires, the accessory should retry the command.

Table 2-15 ACK packet with Command Pending status but no transaction ID

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command: ACK
5	0x06	Command result status: Command Pending
6	0xNN	The ID of the command being acknowledged.
7	0xNN	Maximum amount of time to wait for pending response, in milliseconds (bits 31:24).
8	0xNN	Maximum pending wait, in milliseconds (bits 23:16)
9	0xNN	Maximum pending wait, in milliseconds (bits 15:8)
10	0xNN	Maximum pending wait, in milliseconds (bits 7:0)
11	0xNN	Checksum

Table 2-16 ACK packet with Command Pending status and transaction ID

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0A	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command ID: ACK
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525). This is the same transaction ID that the failed DevDataTransfer packet used.
6	0xNN	transID [bits 7:0]

Byte number	Value	Comment
7	0x06	Command result status: Command Pending
8	0xNN	The ID of the command being acknowledged.
9	0xNN	Maximum amount of time to wait for pending response, in milliseconds (bits 31:24).
10	0xNN	Maximum pending wait, in milliseconds (bits 23:16)
11	0xNN	Maximum pending wait, in milliseconds (bits 15:8)
12	0xNN	Maximum pending wait, in milliseconds (bits 7:0)
13	0xNN	Checksum

If the ACK command returns `SessionWriteFailure` (error code 0x17), the Apple device was unable to send all the data required by a `DevDataTransfer` command. The Apple device adds fields to the ACK packet, as shown in [Table 2-17](#) (page 87), to indicate which `DevDataTransfer` session ID failed and how many bytes were dropped from the end of the packet that the accessory sent. The accessory may assume that the Apple device successfully accepted the earlier bytes.

Note: The `SessionWriteFailure` return is supported only in products running iOS 4 and later versions.

Table 2-17 ACK packet reporting dropped data

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0C	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command ID: ACK
5	0xNN	transID [bits 15:8]: Transaction ID; see " Transaction IDs " (page 525). This is the same transaction ID that the failed <code>DevDataTransfer</code> packet used.
6	0xNN	transID [bits 7:0]
7	0x17	Command result: <code>SessionWriteFailure</code>
8	0x42	The ID of the command being acknowledged (<code>DevDataTransfer</code>).
9	0xNN	sessionID [bits 15:8]: Session ID used by <code>DevDataTransfer</code>
10	0xNN	sessionID [bits 7:0]
11	0xNN	numBytesDropped [bits 31:24]: Number of dropped bytes

Byte number	Value	Comment
12	0xNN	numBytesDropped [bits 23:16]
13	0xNN	numBytesDropped [bits 15:8]
14	0xNN	numBytesDropped [bits 7:0]
15	0xNN	Checksum

Command 0x03: RequestRemoteUIMode

Direction: Accessory to Apple device

The accessory requests the Extended Interface mode from the Apple device. The Apple device responds with "[Command 0x04: ReturnRemoteUIMode](#)" (page 88). This command may be used only if the accessory requests Lingo 0x04 during its identification process.

Table 2-18 RequestRemoteUIMode packet

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x02	Packet payload length
3	0x00	Lingo ID: General lingo
4	0x03	Command ID: RequestRemoteUIMode
5	0xFB	Packet payload checksum byte

Command 0x04: ReturnRemoteUIMode

Direction: Apple device to Accessory

The Apple device reports whether its current operating mode is Extended Interface mode. If the returned mode byte is nonzero (true), the Apple device is in Extended Interface mode; otherwise it is in another mode (standard mode or iPod Out mode). This command may be used only if the accessory requests Lingo 0x04 during its identification process.

Table 2-19 ReturnRemoteUIMode packet

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0x03	Packet payload length
3	0x00	Lingo ID: General lingo
4	0x04	Command ID: ReturnRemoteUIMode
5	0xNN	Mode byte. If nonzero (true), the Apple device is in Extended Interface Mode; if zero (false), it is not.
6	0xNN	Packet payload checksum byte

Command 0x05: EnterRemoteUIMode

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to force it to enter the Extended Interface mode. If the Apple device is already in the Extended Interface mode, it immediately returns a General lingo ACK command packet, notifying the user that the command was successful. This command may be used only if the accessory requests Lingo 0x04 during its identification process.

Note: This command fails if the Apple device does not detect a valid R_{ID} resistor. See “Accessory Detect and Identify” in *MFi Accessory Hardware Specification*.

If the Apple device needs to switch modes, it returns a General lingo ACK Pending command packet, informing the accessory how long it will take the Apple device to switch modes. This is followed by an ACK packet notifying the accessory that the Apple device successfully changed modes. Accessories should honor the timeout returned by the ACK pending before assuming the original command has failed.

If audio is playing when the Apple device enters Extended Interface mode, the Apple device will pause playback. If video is playing, the Apple device will stop playback.

Table 2-20 EnterRemoteUIMode packet

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x02	Packet payload length
3	0x00	Lingo ID: General lingo
4	0x05	Command ID: EnterRemoteUIMode
5	0xF9	Packet payload checksum byte

Command 0x06: ExitRemoteUIMode

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to force it to exit the Extended Interface mode. If the Apple device is already in the Standard UI mode, it immediately returns a General lingo ACK command packet, notifying the user that the command was successful. This command may be used only if the accessory requests Lingo 0x04 during its identification process.

If the Apple device needs to switch modes, it sends a General lingo ACK Pending command packet informing the accessory how long it will take the Apple device to switch modes. This is followed by an ACK packet notifying the accessory that the Apple device successfully changed modes. Accessories should honor the timeout returned by the ACK pending before assuming the original command has failed.

Table 2-21 ExitRemoteUIMode packet

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x02	Packet payload length
3	0x00	Lingo ID; General lingo
4	0x06	Command ID: ExitRemoteUIMode
5	0xF8	Packet payload checksum byte

Command 0x07: RequestiPodName

Direction: Accessory to Apple device

This command retrieves the name of the Apple device. The Apple device responds with a "[Command 0x08: ReturniPodName](#)" (page 91) command containing the name of the Apple device as a null-terminated UTF-8 character array.

Table 2-22 RequestiPodName packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x07	Command: RequestiPodName

Byte number	Value	Comment
5	0xF7	Checksum

Command 0x08: ReturniPodName

Direction: Apple device to Accessory

The Apple device sends this command in response to the "[Command 0x07: RequestiPodName](#)" (page 90) message from the accessory. The Apple device name is encoded as a null-terminated UTF-8 character array. If the Apple device name has not been modified by the user, it is returned as "iPod".

Note: Starting with version 1.02 of the General lingo, the ReturniPodName command on Windows-formatted Apple devices returns the iTunes name of the Apple device instead of the Windows volume name.

Table 2-23 ReturniPodName packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x08	Command: ReturniPodName
5	0xNN...	The name of the Apple device as a null-terminated UTF-8 character array.
(last byte)	0xNN	Checksum

Command 0x09: RequestiPodSoftwareVersion

Direction: Accessory to Apple device

This command retrieves the software version information for the Apple device. The Apple device responds with a "[Command 0x0A: ReturniPodSoftwareVersion](#)" (page 92) containing the major, minor, and revision version numbers.

Note: This command requests the Apple device software version and not the version of a particular lingo protocol.

Table 2-24 RequestiPodSoftwareVersion packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x09	Command: RequestiPodSoftwareVersion
5	0xF5	Checksum

Command 0x0A: ReturniPodSoftwareVersion

Direction: Apple device to Accessory

The Apple device sends this command in response to the "Command 0x09: RequestiPodSoftwareVersion" (page 91) message from the accessory. The Apple device returns each version number as an individual byte, with the major version number sent first. For example, if the major, minor, and revision bytes are returned as 0x01, 0x02, and 0x03, the Apple device software version number is 1.02.03.

Table 2-25 ReturniPodSoftwareVersion packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x0A	Command: ReturniPodSoftwareVersion
5	0xNN	Apple device major version number.
6	0xNN	Apple device minor version number.
7	0xNN	Apple device revision version number.
8	0xNN	Checksum

Command 0x0B: RequestiPodSerialNum

Direction: Accessory to Apple device

This command retrieves the serial number string of the Apple device. The Apple device responds with a "[Command 0x0C: ReturniPodSerialNum](#)" (page 93) containing the serial number as a null-terminated UTF-8 character array.

Table 2-26 RequestiPodSerialNum packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x0B	Command: RequestiPodSerialNum
5	0xF3	Checksum

Command 0x0C: ReturniPodSerialNum

Direction: Apple device to Accessory

The Apple device sends this command in response to the "[Command 0x0B: RequestiPodSerialNum](#)" (page 93) message from the accessory. The Apple device serial number is encoded as a null-terminated UTF-8 character array.

Table 2-27 ReturniPodSerialNum packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x0C	Command: ReturniPodSerialNum
5	0xNN...	The Apple device serial number, as a null-terminated UTF-8 character array.
(last byte)	0xNN	Checksum

Command 0x0F: RequestLingoProtocolVersion

Direction: Accessory to Apple device

This command retrieves version information for any of the lingoes supported by the Apple device. The Apple device responds with a "[Command 0x10: ReturnLingoProtocolVersion](#)" (page 94) containing the major and minor version information of the requested Apple device lingo. This command has one parameter, the lingo whose version information is requested. The Apple device returns an ACK command with a bad parameter status if an accessory calls this command with an invalid or unsupported lingo ID. When an Apple device does not respond to the `GetiPodOptionsForLingo` command, the accessory may use the `RequestLingoProtocolVersion` command to determine what iAP features are available for each lingo used.

Table 2-28 RequestLingoProtocolVersion packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x0F	Command: RequestLingoProtocolVersion
5	0xNN	The lingo for which to request version information.
6	0xNN	Checksum

Command 0x10: ReturnLingoProtocolVersion

Direction: Apple device to Accessory

The Apple device sends this command in response to the "[Command 0x0F: RequestLingoProtocolVersion](#)" (page 94) message from the accessory. The major and minor version information for the requested lingo are returned.

Table 2-29 ReturnLingoProtocolVersion packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x10	Command: ReturnLingoProtocolVersion

Byte number	Value	Comment
5	0xNN	The lingo for which version information is being returned.
6	0xNN	The major protocol version for the given lingo.
7	0xNN	The minor protocol version for the given lingo.
8	0xNN	Checksum

Command 0x11: RequestTransportMaxPayloadSize

Direction: Accessory to Apple device

An accessory sends this command to an Apple device to determine the maximum allowable payload size per packet using the current transport (UART, USB, or USB Host mode). The Apple device replies by sending a `ReturnTransportMaxPayloadSize` command. For the definition of payload size, see ["Command Packet Formats"](#) (page 76).

Table 2-30 RequestTransportMaxPayloadSize packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x00	Lingo ID: General lingo
4	0x11	Command ID: RequestTransportMaxPayloadSize
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	Checksum

Command 0x12: ReturnTransportMaxPayloadSize

Direction: Apple device to Accessory

The Apple device sends this command in response to a `RequestTransportMaxPayloadSize` command, to tell the accessory the maximum allowable packet payload size, in bytes, for the current transport. For the definition of payload size, see ["Command Packet Formats"](#) (page 76)

Table 2-31 ReturnTransportMaxPayloadSize packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x00	Lingo ID: General lingo
4	0x12	Command ID: ReturnTransportMaxPayloadSize
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	maxPayload [bits 15:8]: The maximum allowable packet payload, in bytes.
8	0xNN	maxPayload [bits 7:0]
9	0xNN	Checksum

Command 0x13: IdentifyDeviceLingoes

Direction: Accessory to Apple device

IMPORTANT: New accessory designs must use the Identify Device Preferences and Settings (IDPS) process described in Appendix B, "Accessory Identification" (page 519). This process ensures their compatibility with future firmware. Existing accessory designs may continue to send the IdentifyDeviceLingoes command, identifying the lingoes they support and requesting immediate authentication.

The accessory sends this command to signal its presence and to identify its supported lingoes. In response, the Apple device sends an ACK command. Accessories use the IdentifyDeviceLingoes command to report all supported lingoes; it must be used in place of the Identify (0x01) command.

Note: The IdentifyDeviceLingoes command may also be used to make the Apple device display an "Accessory not supported" message; see the response to Step 6 in Table 2-1 (page 67).

The IdentifyDeviceLingoes command resets all accessory information set by a previous Identify command, including the authentication retry counter and any previously granted authentication access permissions. The payload of this command includes three fields: the Device Lingoes Spoken, Options, and Device ID fields.

Note: An accessory attached via the 30-pin connector must always monitor the Accessory Power output from the Apple device (pin 13 in “Hardware Interfaces” in *MFi Accessory Hardware Specification*). If Accessory Power goes low, even momentarily, the accessory must stop sending iAP commands. After Accessory Power goes high, the accessory must wait at least 80 ms and then send a `StartIDPS` command, following the steps shown in [Table 2-1](#) (page 67) (UART accessories) or [Table 2-2](#) (page 69) (USB or BT accessories).

The `IdentifyDeviceLingoes` command disables all but free lingoes on the current port unless authentication is requested (immediate authentication is also required for Authentication 2.0). For serial ports, this means lingoes 0x00, 0x02, 0x03, and 0x05 may be used, excluding authenticated commands; the USB port will be able to use only the general lingo, 0x00 (see [Table 2-4](#) (page 72)). Accessories that register with this command can use only those lingoes that they specifically identify (see [Table 2-33](#) (page 98)).

If any lingo identified by the `IdentifyDeviceLingoes` command can be used by only one accessory at a time, and that lingo is already in use by a different accessory, the command will fail but no command failure ACK command will be sent to the accessory. The accessory can verify that the `IdentifyDeviceLingoes` command has succeeded by sending a free command with valid parameters and checking the Apple device's response. The Apple device will respond with an ACK response with failure status if any lingo is already in use. An identification failure results in the accessory being able to use only the General lingo (0x00).

This command performs lingo conflict checking to ensure that single-instance lingoes, such as Display Remote, are used on only one port at a time. If the `IdentifyDeviceLingoes` command is acknowledged with a status of 0x15 (Maximum number of accessory connections already reached), the accessory must not send any further iAP traffic until it has been disconnected and reconnected to the Apple device.

For sample command sequences in which an accessory identifies its supported lingoes, using `IdentifyDeviceLingoes`, see [Sample Identification Sequences](#) (page 402).

Table 2-32 `IdentifyDeviceLingoes` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0E	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x13	Command: <code>IdentifyDeviceLingoes</code>
5-8	0xNN	Device Lingoes Spoken; see Table 2-33 (page 98).
9-12	0xNN	Options; see Table 2-34 (page 98).
13-16	0xNN	Device ID. Accessories must send a unique identifier, supplied by the iPod Authentication Coprocessor, if they require authentication. If an accessory does not require authentication, it can send the Device ID to 0x00000000 and set the authentication option bits to 0x0.
17	0xNN	Checksum

Use the Device Lingo Spoken field as a bit field to set each bit corresponding to the lingo supported by the accessory. For example, if an accessory supports both the Simple Remote and Accessory Power lingoes, the value of the bit field is 0x00000025 and the low byte in binary is 00100101.

Table 2-33 Device Lingo Spoken bits

Bit	Supported lingo
0	General lingo (must be set by all accessories)
1	Reserved; set to 0
2	Simple Remote lingo
3	Display Remote lingo
4	Extended Interface lingo
5	Accessory Power lingo
6	USB Host Mode lingo
7	RF Tuner lingo
8	Accessory Equalizer lingo
9	Sports lingo
10	Digital Audio lingo
11	Reserved; set to 0
12	Storage lingo
31:13	Reserved; set to 0

Note: The iPod Out lingo (Lingo 0x0D) and the Location lingo (Lingo 0x0E) are available only if the accessory has identified itself using IDPS. See ["Accessory Signaling and Initialization"](#) (page 66).

The bits of the Options field are defined as shown in [Table 2-34](#) (page 98).

Table 2-34 IdentifyDeviceLingo Options bits

Bits	Meaning
1:0	Authentication control bits. These bits have the following meanings: 00 = no authentication is supported or required 01 = defer authentication until an authenticated command is used (Authentication 1.0 only); see Note below 10 = authenticate immediately after identification (required for Authentication 2.0). 11 = reserved

Bits	Meaning
3:2	Power control bits. These bits have the following meanings: 00 = low power mode, as defined in the section “Accessory Power Policy” in <i>MFi Accessory Hardware Specification</i> 01 = intermittent high power; the accessory requires up to 100mA maximum during playback operation 10 = reserved 11 = constant high power; the accessory requires more than low power from the Apple device (up to 100 mA maximum) at all times. This mode must not be declared unless the accessory provides power as specified in “Supplying USB Power” in <i>MFi Accessory Hardware Specification</i> .
31:4	Reserved; set to 0

Note: Certain lingoes require immediate authentication. Requesting deferred authentication with the RF Tuner, Accessory Equalizer, and Digital Audio lingoes results in a command failed ACK return from the Apple device.

Accessories identifying using the `IdentifyDeviceLingoes` command receive notifications when the Apple device changes state. See “[Command 0x23: NotifyiPodStateChange](#)” (page 109) for more details.

Note: If an accessory uses an invalid Device ID during an identification attempt, the Apple device returns a bad parameter error (0x04) ACK. If the accessory claims a Device Lingo Spoken that is not supported by the attached Apple device, the Apple device returns a command failed (0x02) ACK.

Cancelling a Current Authentication Process With `IdentifyDeviceLingoes`

For best success calling `IdentifyDeviceLingoes`, use the following sequence:

1. For UART-based communications, ensure that the Accessory Identify pin (pin 10) is connected to the correct resistors and that the Accessory Detect (pin 20) pin is grounded; see “Accessory Detect and Identify” in *MFi Accessory Hardware Specification*. Also, precede all iAP packets by an extra 0xFF autobaud synchronization byte.
2. The accessory sends an “artificial” `IdentifyDeviceLingoes` command with the lingo mask set to only the General lingo, the option bitmask set to 0x0 (no options), and the Device ID set to 0x0. This will cancel any active authentication process on the current Apple device accessory port.
3. The accessory waits up to 500 ms for the Apple device to send an ACK success response. If this does not happen within 500 ms, the accessory may return to Step 2 as many times as desired.
4. The Apple device follows its ACK success response by sending a `GetAccessoryInfo` command with a valid Accessory Info Type parameter.
5. The accessory must respond with a `RetAccessoryInfo` command containing the requested information. This information will be superseded later when the accessory sends `RetAccessoryInfo` again during authentication. See [Table 2-5](#) (page 74).

6. The accessory now sends a “genuine” `IdentifyDeviceLingoes` command, passing the correct parameters, and proceeds with identification and authentication.

Sample command sequences that illustrate some of these best practices are listed in [Table F-41](#) (page 621) and [Table F-42](#) (page 622).

Command 0x14: GetDevAuthenticationInfo

Direction: Apple device to Accessory

The Apple device sends this command to obtain authentication information from the accessory. The command is sent only if the accessory has indicated that it supports authentication during its identification process and has passed a valid, nonzero Device ID. In response, the accessory sends "[Command 0x15: RetDevAuthenticationInfo](#)" (page 100).

Table 2-35 GetDevAuthenticationInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x14	Command: GetDevAuthenticationInfo
5	0xEA	Checksum

Command 0x15: RetDevAuthenticationInfo

Direction: Accessory to Apple device

The accessory indicates the iAP authentication version that it supports by returning this command in response to a "[Command 0x14: GetDevAuthenticationInfo](#)" (page 100) command from the Apple device. The authentication version returned by this command must be consistent with the Device ID sent during its identification process.

The returned packet may have either of two formats, depending on whether the authentication process is Authentication 1.0 or 2.0. See [Table 2-36](#) (page 100) and [Table 2-37](#) (page 101).

Table 2-36 RetDevAuthenticationInfo packet, Authentication 1.0

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x15	Command: RetDevAuthenticationInfo
5	0xNN	Authentication protocol major version number
6	0xNN	Authentication protocol minor version number
7	0xNN	Checksum

Table 2-37 RetDevAuthenticationInfo packet, Authentication 2.0

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x15	Command: RetDevAuthenticationInfo
5	0x02	Authentication major version (0x02)
6	0x00	Authentication minor version (0x00)
7	0xNN	X.509 certificate current section index (0 = first section, 1 = second section, and so on)
8	0xNN	X.509 certificate maximum section index (0 = one section total, 1 = two sections, and so on)
9-NN	0xNN...	X.509 certificate data. If the data length exceeds 500 bytes it must be broken into sections, each not more than 500 bytes.
(last byte)	0xNN	Checksum

Note: During Authentication 2.0, not more than 500 certificate bytes may be sent at once. If the X.509 certificate is larger than 500 bytes, the accessory must divide it into sections, each not larger than 500 bytes, for reassembly by the Apple device. The Apple device will send an ACK command for each certificate section up to, but not including, the last one. The accessory must wait for the ACK command before sending the next certificate section. The final certificate section is acknowledged by an `AckDevAuthenticationInfo` command.

The packet format shown in [Table 2-37](#) (page 101) can deliver only 249 bytes of certificate data. To deliver more data per packet, up to 500 bytes, use the large packet format specified in ["Large Packet Format"](#) (page 77).

The ACK response to each certificate section only acknowledges receipt. When all the sections have been assembled, the Apple device evaluates the whole certificate and sends the `AckDevAuthenticationInfo` command that states whether or not it is valid.

Command 0x16: AckDevAuthenticationInfo

Direction: Apple device to Accessory

The Apple device sends this command in response to ["Command 0x15: RetDevAuthenticationInfo"](#) (page 100). It indicates the current state of the accessory authentication information. If the accessory receives a nonzero status, the Apple device does not support the accessory's authentication version and the accessory will fail authentication.

Note: The accessory must send certificate data in sections not larger than 500 bytes, for reassembly by the Apple device, and must wait for an ACK command after each one. The Apple device acknowledges the final certificate section with this command.

Table 2-38 AckDevAuthenticationInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x16	Command: AckDevAuthenticationInfo
5	0xNN	Status of authentication information. Possible values are: 0x00 = Authentication information supported 0x04 = Certificate too long or sections out of order 0x08 = Authentication information unsupported 0x0A = Certificate is invalid 0x0B = Certificate permissions are invalid
6	0xNN	Checksum

Command 0x17: GetDevAuthenticationSignature

Direction: Apple device to Accessory

The Apple device sends this command to authenticate an accessory that has identified itself as requiring authentication. Authentication occurs either immediately upon identification or when the accessory attempts to use a restricted lingo or command. The accessory calculates its digital signature based on the challenge offered by the Apple device and sends the results back to the Apple device using "Command 0x18: RetDevAuthenticationSignature" (page 103).

The authentication retry counter is used to track the number of retries. If the returned signature cannot be verified, the Apple device responds with a nonzero "Command 0x19: AckDevAuthenticationStatus" (page 104), followed immediately by another "Command 0x17: GetDevAuthenticationSignature" (page 103).

The retry counter is set to 0x01 for the first authentication attempt and incremented each time the Apple device retries the `GetDevAuthenticationSignature` command. Accessories using Authentication 1.0 are allowed up to four retries and a maximum of 30 seconds (up to 7.5 seconds per retry) for the authentication process. Accessories using Authentication 2.0 are allowed up to two retries and a maximum of 150 seconds (up to 75 seconds per retry) for the authentication process. If an accessory fails to respond to the `GetDevAuthenticationSignature` command, the authentication process will fail and the accessory will not be able to use any of the authenticated commands.

Table 2-39 GetDevAuthenticationSignature packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x13 or 0x17	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x17	Command: <code>GetDevAuthenticationSignature</code>
5–20 or 5–24	0xNN...	An offered challenge sent for the accessory to sign and return (16 bytes for Authentication 1.0, 20 bytes for Authentication 2.0).
NN	0xNN	Authentication retry counter. The authentication process terminates after the maximum retry count or maximum timeout interval has been reached, whichever comes first. When the authentication process fails, only nonauthenticated lingoes and commands are usable.
(last byte)	0xNN	Checksum

Command 0x18: RetDevAuthenticationSignature

Direction: Accessory to Apple device

The accessory sends this command to the Apple device in response to "[Command 0x17: GetDevAuthenticationSignature](#)" (page 103). The Apple device verifies the digital signature, calculated by the accessory based on the offered challenge. If verification passes, the Apple device authenticates the accessory and updates its lingo and command access permissions accordingly. The authentication status is sent to the accessory using "[Command 0x19: AckDevAuthenticationStatus](#)" (page 104).

Table 2-40 RetDevAuthenticationSignature packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x18	Command: RetDevAuthenticationSignature
5	0xNN...	The digital signature calculated by the accessory, based on the offered challenge (variable length).
(last byte)	0xNN	Checksum

Command 0x19: AckDevAuthenticationStatus

Direction: Apple device to Accessory

The Apple device sends this command to the accessory in response to the "[Command 0x18: RetDevAuthenticationSignature](#)" (page 103) command. It indicates the current accessory authentication state. If the accessory receives a nonzero status, the accessory has failed authentication and will only be able to use unauthenticated lingo commands.

If the accessory receives a zero status, the Apple device has successfully authenticated the accessory. The accessory may then use the requested authenticated lingoes and commands. Optionally, the accessory may begin the process of authenticating the Apple device, by sending "[Command 0x1A: GetiPodAuthenticationInfo](#)" (page 105).

Table 2-41 AckDevAuthenticationStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x19	Command: AckDevAuthenticationStatus

Byte number	Value	Comment
5	0xNN	Status of the authentication operation: 0x00 = Authentication operation passed All other values = Authentication operation failed; see Table 2-14 (page 85)
6	0xNN	Checksum

Command 0x1A: GetiPodAuthenticationInfo

Direction: Accessory to Apple device

The accessory sends this command to obtain authentication information from the Apple device. The accessory should send this command only if the accessory has indicated that it supports authentication during its identification process and the Apple device has successfully authenticated the accessory. (Accessory authentication is successful when the accessory receives the "[Command 0x19: AckDevAuthenticationStatus](#)" (page 104) command with a status of 0x00). In response to GetiPodAuthenticationInfo the Apple device sends "[Command 0x1B: RetiPodAuthenticationInfo](#)" (page 105).

Table 2-42 GetiPodAuthenticationInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1A	Command: GetiPodAuthenticationInfo
5	0xE4	Checksum

Command 0x1B: RetiPodAuthenticationInfo

Direction: Apple device to Accessory

The Apple device returns this command in response to "[Command 0x1A: GetiPodAuthenticationInfo](#)" (page 105) from the accessory.

Table 2-43 RetiPodAuthenticationInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1B	Command: <code>RetiPodAuthenticationInfo</code>
5	0x02	Authentication major version (0x02)
6	0x00	Authentication minor version (0x00)
7	0xNN...	X.509 certificate current section index (0 = first section, 1 = second section, and so on)
8	0xNN	X.509 certificate maximum section index (0 = one section total, 1 = two sections, and so on)
9-NN	0xNN	X.509 certificate data. If the data length exceeds 249 bytes it will be broken into sections, each not more than 500 bytes, and may be sent in the large packet format specified in " Large Packet Format " (page 77).
(last byte)	0xNN	Checksum

Note: Authentication version 2.00 (major version 0x02, minor version 0x00) is the only version supported by Apple devices that can be authenticated by accessories.

Command 0x1C: `AckIPodAuthenticationInfo`

Direction: Accessory to Apple device

The accessory sends this command to the Apple device in response to "[Command 0x1B: `RetiPodAuthenticationInfo`](#)" (page 105). It indicates the current state of the Apple device authentication information version. If the accessory sends a nonzero status, it indicates that it will not be able to authenticate the Apple device due to an invalid X.509 certificate.

Table 2-44 `AckIPodAuthenticationInfo` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1C	Command: <code>AckIPodAuthenticationInfo</code>

Byte number	Value	Comment
5	0xNN	Status of the authentication information: 0x00 = authentication information valid Other values = authentication information not valid
6	0xNN	Checksum

Command 0x1D: GetiPodAuthenticationSignature

Direction: Accessory to Apple device

The accessory uses this command to send an offered challenge to the Apple device for digital signature. In response, the Apple device returns its signed challenge to the accessory using "[Command 0x1E: RetiPodAuthenticationSignature](#)" (page 107). Accessories must implement the authentication retry feature described in "[Command 0x17: GetDevAuthenticationSignature](#)" (page 103), allowing the Apple device two retries in 150 seconds. The retry counter must be set to 0x01 in the first `GetiPodAuthenticationSignature` command sent to the Apple device and must be incremented for each subsequent attempt. Authentication must fail after either the retry count or maximum response interval have been exceeded since the first `GetiPodAuthenticationSignature` command was sent.

Table 2-45 `GetiPodAuthenticationSignature` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1D	Command: <code>GetiPodAuthenticationSignature</code>
5–24	0xNN...	An offered challenge for the Apple device to sign and return (20 bytes)
25	0x00	Authentication retry counter. A maximum of two retries or 150 seconds, whichever happens first, should occur before the authentication process is terminated.
26	0xNN	Checksum

Command 0x1E: RetiPodAuthenticationSignature

Direction: Apple device to Accessory

The Apple device sends this command to the accessory in response to "Command 0x1D: [GetiPodAuthenticationSignature](#)" (page 107). The accessory verifies the digital signature, calculated by the Apple device based on the offered challenge, and, if verification passes, authenticates the Apple device. The accessory sends the authentication status to the Apple device using "Command 0x1F: [AckiPodAuthenticationStatus](#)" (page 108).

Table 2-46 RetiPodAuthenticationSignature packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1E	Command: RetiPodAuthenticationSignature
5 ... N	0xNN...	The digital signature calculated by the Apple device
(last byte)	0xNN	Checksum

Command 0x1F: AckiPodAuthenticationStatus

Direction: Accessory to Apple device

The accessory sends this command to the Apple device in response to "Command 0x1E: [RetiPodAuthenticationSignature](#)" (page 107). It indicates the current Apple device authentication state. The accessory should return a nonzero ACK for each failed authentication attempt.

Table 2-47 AckiPodAuthenticationStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1F	Command: AckiPodAuthenticationStatus
5	0xNN	Status of authentication information: 0x00 = authentication operation passed Other values = authentication operation failed
6	0xNN	Checksum

Command 0x23: NotifyPodStateChange

Direction: Apple device to Accessory

When the Apple device power state is about to change, the Apple device sends this notification command to accessories that identify using IDPS or `IdentifyDeviceLingo`s. If the accessory identifies using the deprecated command `Identify`, this notification is not sent. The state change byte indicates the specific Apple device state transition. If the Apple device is switching from a Power On state to a Sleep state, accessories must immediately reduce their power consumption to low power mode, as defined in the section “Accessory Power Policy” in *MFi Accessory Hardware Specification*. When the Apple device has transitioned to Hibernate state, self-powered accessories are expected to automatically reidentify themselves 80 ms after current is restored on the Accessory Power line of the 30-pin connector.

Table 2-48 NotifyPodStateChange packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x23	Command: NotifyPodStateChange
5	0xNN	StateChg (1 byte). The Apple device state change. Possible values are: 0x00 = reserved. 0x01 = current from the Apple device on the Accessory Power line of the 30-pin connector going to Hibernate state (no power) and not preserving menu selections or playback context (see “Power States” in <i>MFi Accessory Hardware Specification</i>). 0x02 = current from the Apple device going to Hibernate state (no power) but preserving menu selections and playback context. 0x03 = current from the Apple device going to Sleep state (drawing less than 5 mA current). 0x04 = current from the Apple device going to the Power On state. 0x05–0xFF = reserved.
6	0xNN	Checksum

Note: Firmware version 1.0.0 of the iPod nano reported its state change incorrectly. The `StateChg` byte is decremented by 1, so that 0x00 represents a value of 0x01, 0x01 represents a value of 0x02, and so forth. All later versions of the nano firmware conform to the table above.

Command 0x24: GetiPodOptions

Direction: Accessory to Apple device

IMPORTANT: The preferred way to obtain information about the capabilities of an attached Apple device is to send "Command 0x4B: GetiPodOptionsForLingo" (page 165). New accessory designs must send `GetiPodOptionsForLingo` first. If and only if an ACK command with Bad Parameter status (0x04) is returned, then the accessory may send `GetiPodOptions` instead.

The accessory sends this command to ask the Apple device to return a 64-bit field that defines the options that the Apple device supports. In response, the Apple device sends a `RetiPodOptions` command.

Table 2-49 `GetiPodOptions` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x00	Lingo ID: General lingo
4	0x24	Command ID: <code>GetiPodOptions</code>
5	0xDA	Checksum

Command 0x25: RetiPodOptions

Direction: Apple device to Accessory

The Apple device sends this command in response to a `GetiPodOptions` command from the accessory.

Table 2-50 `RetiPodOptions` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0A	Length of packet
3	0x00	Lingo ID: General lingo

Byte number	Value	Comment
4	0x25	Command ID: <code>RetiPodOptions</code>
5	0xNN	Option bits (bits 63:56): Reserved
6	0xNN	Option bits (bits 55:48): Reserved
7	0xNN	Option bits (bits 47:40): Reserved
8	0xNN	Option bits (bits 39:32): Reserved
9	0xNN	Option bits (bits 31:24): Reserved
10	0xNN	Option bits (bits 23:16): Reserved
11	0xNN	Option bits (bits 15:8): Reserved
12	0xNN	Option bits (bits 7:0): Bit 1: the Apple device supports using <code>SetiPodPreferences</code> to control line-out usage Bit 0: the Apple device supports video output
13	0xNN	Checksum

Command 0x27: `GetAccessoryInfo`

Direction: Apple device to Accessory

The Apple device sends this command to accessories that identify themselves using the `IdentifyDeviceLingoes` command to obtain certain information from the accessory. The accessory must respond with a `RetAccessoryInfo` command for the required accessory Info Types. The Apple device will use the information gathered to:

- Display accessory information in the Settings>About box on the Apple device
- Display a message to the user if the Apple device firmware needs to be updated to support the accessory
- Display a message to the user if the Apple device firmware does not support the accessory

The `GetAccessoryInfo` command sends the Accessory Info Type and the Accessory Info Type parameters to the accessory. [Table 2-51](#) (page 112) lists the number of parameter bytes for the corresponding Accessory Info Types. The Apple device requests each of the Accessory Info Types in the order in which they appear in [Table 2-51](#) (page 112). Because the Accessory minimum supported Apple device firmware version request (which sends the Apple device model number as a parameter) is sent before any Accessory minimum supported lingo version requests, the accessory has the option of changing its responses to those appropriate for the Apple device model.

When the `GetAccessoryInfo` command is sent with the accessory minimum supported lingo version info type, the 1-byte lingo number for which the Apple device is requesting the minimum supported version is sent as a parameter. The Apple device will send the `GetAccessoryInfo` command with this Accessory Info Type for every lingo that the accessory indicates it supports.

The Apple device begins sending `GetAccessoryInfo` commands as soon as an accessory identifies itself successfully via the `IdentifyDeviceLingoes` command and either enters the background authentication state or does not request authentication. If the accessory does not respond, the Apple device waits 5 seconds for a response before timing out and retrying. It retries a maximum of three times.

The types of data that `GetAccessoryInfo` can request are listed in [Table 2-51](#) (page 112). The command's packet has two formats, depending on the data type, as shown in the table.

If an accessory supports `GetAccessoryInfo` requests from the Apple device (Info Type 0x0B), it must register its support during the IDPS process by sending the Apple device an `AccInfoToken` with an `accInfoType` of 0x0B; see [Table 2-81](#) (page 135).

Table 2-51 Accessory Info Type values

Value	Meaning	Parameter bytes	Requirement	Packet format
0x00	Accessory info capabilities	0	Required	Table 2-52 (page 112)
0x01	Accessory name	0	Required	Table 2-52 (page 112)
0x02	Accessory minimum supported iPod firmware version (deprecated)	7	Optional	Table 2-53 (page 113)
0x03	Accessory minimum supported lingo version	1	Optional	Table 2-54 (page 113)
0x04	Accessory firmware version	0	Required	Table 2-52 (page 112)
0x05	Accessory hardware version	0	Required	Table 2-52 (page 112)
0x06	Accessory manufacturer	0	Required	Table 2-52 (page 112)
0x07	Accessory model number	0	Required	Table 2-52 (page 112)
0x08	Accessory serial number	0	Optional	Table 2-52 (page 112)
0x09	Accessory incoming maximum payload size	0	Optional	Table 2-52 (page 112)
0x0A	Reserved			
0x0B	Accessory status types supported; see Table 2-64 (page 120)	0	Optional	Table 2-52 (page 112)
0x0C-0xFF	Reserved			

Table 2-52 `GetAccessoryInfo` packet with Accessory Info Type = 0x00-0x01, 0x04-0x09, or 0x0B

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet

Byte number	Value	Comment
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x27	Command ID: GetAccessoryInfo
5	0xNN	Accessory Info Type; see Table 2-51 (page 112)
6	0xNN	Checksum

Table 2-53 GetAccessoryInfo packet with Accessory Info Type = 0x02 (deprecated)

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x0A	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x27	Command ID: GetAccessoryInfo
5	0x02	Accessory Info Type (see Table 2-51 (page 112))
6	0xNN	Apple device Model ID (bits 31:24)
7	0xNN	Apple device Model ID (bits 23:16)
8	0xNN	Apple device Model ID (bits 15:8)
9	0xNN	Apple device Model ID (bits 7:0)
10	0xNN	Apple device firmware major version number
11	0xNN	Apple device firmware minor version number
12	0xNN	Apple device firmware revision version number
13	0xNN	Checksum

Table 2-54 GetAccessoryInfo packet with Accessory Info Type = 0x03

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x04	Length of packet payload

Byte number	Value	Comment
3	0x00	Lingo ID: General lingo
4	0x27	Command ID: GetAccessoryInfo
5	0x03	Accessory Info Type (see Table 2-51 (page 112))
6	0xNN	Lingo ID
7	0xNN	Checksum

Command 0x28: RetAccessoryInfo

Direction: Accessory to Apple device

If the accessory is attached to a non-IDPS Apple device, it can reply to a second `GetAccessoryInfo` request with an `IdentifyDeviceLingoes` command, as shown in Step 10 of [Table F-41](#) (page 621).

Except for that case, the accessory must reply with this command to every receipt of `GetAccessoryInfo` from the Apple device. It should reply within 500 ms. The data contained in the packet must be responsive to the Accessory Info Type requested by the Apple device.

When the `RetAccessoryInfo` command is returning the accessory info capabilities, a bit-field is returned where every set bit represents a supported Accessory Info Type. See [Table 2-51](#) (page 112) for a description of the Accessory Info Type; see [Table 2-55](#) (page 114) for the meanings of its returned data bytes. Each Accessory Info Type value must be returned in the appropriate `RetAccessoryInfo` packet format, as shown in the table.

Table 2-55 Accessory Info Type values for `RetAccessoryInfo`

Value	Meaning	Parameter bytes	Packet format
0x00	Accessory info capabilities	4	Table 2-56 (page 115)
0x01	Accessory name	1–64	Table 2-58 (page 116)
0x02	Accessory minimum supported iPod firmware version (deprecated)	7	Table 2-59 (page 117)
0x03	Accessory minimum supported lingo version	3	Table 2-60 (page 118)
0x04	Accessory firmware version	3	Table 2-61 (page 118)
0x05	Accessory hardware version	3	Table 2-61 (page 118)
0x06	Accessory manufacturer	1–64	Table 2-58 (page 116)
0x07	Accessory model number	1–64	Table 2-58 (page 116)
0x08	Accessory serial number	1–64	Table 2-58 (page 116)
0x09	Accessory incoming maximum payload size	2	Table 2-62 (page 119)

Value	Meaning	Parameter bytes	Packet format
0x0A	Reserved	N/A	
0x0B	Accessory status types supported; see Table 2-64 (page 120)	4	Table 2-63 (page 119)
0x0C-0xFF	Reserved	N/A	

Table 2-56 RetAccessoryInfo packet with Accessory Info Type = 0x00

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x07	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0x00	Accessory Info Type. See Table 2-55 (page 114)
6	0xNN	Accessory capabilities, bits 31:24; see Table 2-57 (page 115))
7	0xNN	Accessory capabilities, bits 23:16
8	0xNN	Accessory capabilities, bits 15:8
9	0xNN	Accessory capabilities, bits 7:0
10	0xNN	Checksum

Table 2-57 Accessory info capabilities bit field

Bit	Capability supported	Accessory requirement	Notes
0	Accessory info capabilities	Required; set to 1	
1	Accessory name	Required; set to 1	Note 1
2	Accessory minimum supported Apple device firmware version	Optional	
3	Accessory minimum supported lingo version	Optional	
4	Accessory firmware version	Required; set to 1	
5	Accessory hardware version	Required; set to 1	
6	Accessory manufacturer	Required; set to 1	Note 1
7	Accessory model number	Required; set to 1	Note 1

Bit	Capability supported	Accessory requirement	Notes
8	Accessory serial number	Optional	Note 1
9	Accessory incoming maximum payload size	Optional	
10	Reserved		
11	Accessory status notifications	Optional	
12-17	Reserved		
18	Asynchronous playback state changes	Optional	Note 2
19-31	Reserved		

Notes:

1. The accessory name, manufacturer, model number, and serial number are sent as UTF-8 character arrays and must be less than or equal to 64 bytes (including a null termination character). See [Table 2-57](#) (page 115) for details. Even when this condition is met, the Apple device may not be capable of displaying all the characters in the array in its About box.
2. An accessory should set Bit 18 to 1 if it is able to handle Apple device-driven changes in playback state or playlist contents; see "[Playback Status Notifications](#)" (page 418). Such accessories may operate alongside other accessories connected to a single Apple device. All Bluetooth accessories must support this capability.

IMPORTANT: Accessories must provide capability information for all required items listed in [Table 2-57](#) (page 115).

Table 2-58 RetAccessoryInfo packet with Accessory Info Type = 0x01/0x06/0x07/0x08

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0xNN	Accessory Info Type. See Table 2-55 (page 114)
6...	0xNN...	Null-terminated UTF-8 character array
(last byte)	0xNN	Checksum

The `GetAccessoryInfo` packet with Accessory Info Type = 0x02 is deprecated, and is sent to an accessory only by older iPods; nevertheless, the accessory should reply. When the accessory returns the 3-byte minimum supported Apple device firmware major/minor/revision version requested, it must also return the 4-byte Apple device model ID. The accessory should therefore store all the model numbers of older iPods that support it, along with their corresponding minimum supported Apple device firmware versions.

If an unknown or unsupported Apple device model ID is sent to the accessory, the accessory should return the `RetAccessoryInfo` command with the Apple device Model ID and 0xFF as the Apple device firmware major/minor/revision version numbers.

Table 2-59 `RetAccessoryInfo` packet with Accessory Info Type = 0x02 (deprecated)

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x0A	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: <code>RetAccessoryInfo</code>
5	0x02	Accessory Info Type. See Table 2-55 (page 114)
6	0xNN	Apple device Model ID (bits 31:24)
7	0xNN	Apple device Model ID (bits 23:16)
8	0xNN	Apple device Model ID (bits 15:8)
9	0xNN	Apple device Model ID (bits 7:0)
10	0xNN	minimum supported Apple device firmware major version
11	0xNN	minimum supported Apple device firmware minor version
12	0xNN	minimum supported Apple device firmware revision version
13	0xNN	Checksum

If the accessory's minimum supported Apple device firmware version is higher than the Apple device firmware version, and one or more of the lingo version numbers is higher than that supported by the Apple device, the Apple device will display a message to the user indicating that the Apple device firmware should be updated.

If the accessory's minimum supported Apple device firmware version is smaller than or equal to the Apple device firmware version or any of the major/minor/revision numbers are 0xFF, and one or more of the lingo version numbers is higher than that supported by the Apple device, the Apple device will display a message to the user indicating that the Apple device does not support the accessory.

Table 2-60 RetAccessoryInfo packet with Accessory Info Type = 0x03

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0x03	Accessory Info Type. See Table 2-55 (page 114)
6	0xNN	Lingo ID
7	0xNN	Major protocol version for lingo ID
8	0xNN	Minor protocol version for lingo ID
9	0xNN	Checksum

Table 2-61 RetAccessoryInfo packet with Accessory Info Type = 0x04/0x05

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0xNN	Accessory Info Type. See Table 2-55 (page 114)
6	0xNN	Accessory major version number
7	0xNN	Accessory minor version number
8	0xNN	Accessory revision version number
9	0xNN	Checksum

The accessory's incoming maximum packet size indicates the maximum size of a packet from the Apple device that the accessory can support. If the accessory does not return a value, the Apple device assumes that the maximum packet size is 1024 bytes. The maximum packet size must be bigger or equal to 128 bytes and smaller or equal to 65535 (0xFFFF) bytes.

Only Apple device packets that contain a UTF-8 character array can be larger than maximum packet size. In that case, the Apple device will insert a null termination character into the UTF-8 array to force it to fit into the packet. This does not apply to commands that allow multipacket responses.

Note: The maximum packet size restriction does not apply to General lingo packets related to authentication.

Table 2-62 RetAccessoryInfo packet with Accessory Info Type = 0x09

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0x09	Accessory Info Type. See Table 2-55 (page 114)
6	0xNN	Max payload size (bits 15:8)
7	0xNN	Max payload size (bits 7:0)
8	0xNN	Checksum

Table 2-63 RetAccessoryInfo packet with Accessory Info Type = 0x0B

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x07	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0x0B	Accessory Info Type. See Table 2-55 (page 114)
6	0xNN	Accessory status types supported (bits 31:24); see Table 2-64 (page 120).
7	0xNN	Accessory status types supported (bits 23:16)
8	0xNN	Accessory status types supported (bits 15:8)
9	0xNN	Accessory status types supported (bits 7:0)
10	0xNN	Checksum

Table 2-64 Accessory status values

Mask bit	Status type	Status description
01:00	Reserved	
02	0x02	Fault condition; see "Command 0x48: AccessoryStatusNotification" (page 154) for fault condition parameters.
31:03	Reserved	

Command 0x29: GetiPodPreferences

Direction: Accessory to Apple device

The accessory sends this command to ask the Apple device to return a specific class of preferences set on it, as defined by a preference class ID. In response, the Apple device sends a `RetiPodPreferences` command.

Table 2-65 GetiPodPreferences packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x00	Lingo ID: General lingo
4	0x29	Command ID: GetiPodPreferences
5	0xNN	Preference class ID (see Table 2-66 (page 120)). Class IDs 0x00-0x02, 0x08-0x0A, and 0x0C-0x0D are available only if the Apple device supports video playback; see "Video Output Settings" (page 48). Class ID 0x03 is available only if the Apple device supports line-out usage.
6	0xNN	Checksum

Table 2-66 Apple device preference class and setting IDs

Class ID	Preference	Setting	Setting ID	Description
0x00	Video out setting	Off	0x00	Disables Apple device video output of any kind.
		On	0x01	Enables Apple device video output based on the other video preferences (NTSC/PAL, screen configuration, etc.).
		Reserved	0x02-0xFF	

Class ID	Preference	Setting	Setting ID	Description
0x01	Screen configuration	Fill entire screen	0x00	Expand the video image to fill the entire screen without letterbox or pillarbox black bars. The “square” pixel proportions are preserved by enlarging both vertical and horizontal dimensions by the same percentage. Depending on the Apple device’s media source and video monitor destination aspect ratios, this may result in cutting off the top and bottom or left and right edges of the image; see Note 1, below.
		Fit to screen edge	0x01	Expand the video image to screen edge. The Apple device enlarges the source media so that its top and bottom or left and right edges end at the screen edge without losing any vertical or horizontal image information. The “square” pixel vertical and horizontal proportions are preserved by being enlarged by the same percentage. Depending on the Apple device’s media source and video monitor destination aspect ratio, this may result in adding either letterbox black bars at the top and bottom of the screen or pillarbox black bars at the left and right of the screen; see Note 2, below.
		Reserved	0x02–0xFF	
0x02	Video signal format	NTSC	0x00	NTSC video format and timing.
		PAL	0x01	PAL video format and timing.
		Reserved	0x02–0xFF	
0x03	Line-out usage	Not used	0x00	Line out is not used by accessory and may be disabled by the Apple device. This setting may also indicate that the USB digital audio output routing has been selected. See Note 3, below, for requirements and restrictions.
		Used	0x01	Line out is used by accessory and is enabled by the Apple device. This setting may also indicate that the analog line out audio routing has been selected instead of the USB digital audio routing. See Note 3, below, for requirements and restrictions.
		Reserved	0x02–0xFF	
0x04–0x07	Reserved			
0x08	Video-out connection	None	0x00	No change to current connection.

Class ID	Preference	Setting	Setting ID	Description
		Composite	0x01	Composite video connection (interlaced video only).
		S-Video	0x02	S-Video video connection (interlaced video only).
		Component	0x03	Component Y/Pr/Pb video connection (interlaced or progressive scan, depending on the Apple device model).
		Reserved	0x04-0xFF	
0x09	Closed captioning	Off	0x00	Closed caption signaling is disabled.
		On	0x01	The Apple device overlays closed caption text on the video content before outputting the video signal. The text is not present on line 21, so the video monitor cannot do any closed caption processing. Note: Closed captions and subtitles cannot be enabled at the same time.
		Reserved	0x02-0xFF	
0x0A	Video monitor aspect ratio (can be set only while video is paused or not playing)	4:3 aspect ratio (fullscreen)	0x00	Used when the video output destination monitor has a horizontal to vertical aspect ratio of 4 to 3, such as an original television monitor format. Depending on the media source format and screen configuration preference, the media content may be displayed fullscreen or have letterbox bars. Apple devices that support widescreen signaling use it to send aspect ratio information to the monitor.
		16:9 aspect ratio (widescreen)	0x01	Used when the video output destination monitor has a horizontal to vertical aspect ratio of 16 to 9, such as the widescreen theater format. Depending on the media source format and screen configuration preference, the media content may be displayed widescreen or have pillarbox bars. Apple devices that support widescreen signaling use it to send aspect ratio information to the monitor.
		Reserved	0x02-0xFF	
0x0B	Reserved			
0x0C	Subtitles (see Note 4, below)	Subtitles off	0x00	Subtitle overlays are disabled.

Class ID	Preference	Setting	Setting ID	Description
		Subtitles on	0x01	The Apple device overlays subtitle text on the video content before outputting the video signal. The video monitor does not need to do any subtitle processing. Note: Closed captions and subtitles cannot be enabled at the same time.
		Reserved	0x02-0xFF	
0x0D	Video alternate audio channel	Alternate audio off	0x00	The alternate audio channel is disabled.
		Alternate audio on	0x01	The alternate audio channel is enabled.
		Reserved	0x02-0xFF	
0x0E	Reserved			
0x0F	Pause on power removal	Pause off	0x00	The Apple product does not pause its playback when USB power is removed.
		Pause on	0x01	The Apple product pauses its playback when USB power is removed.
		Reserved	0x02-0xFF	
0x10-0x13	Reserved			
0x14	Accessibility preference	VoiceOver off	0x00	VoiceOver is disabled.
		VoiceOver on	0x01	VoiceOver is enabled.
		Reserved	0x02-0xFF	
0x15-0xFF	Reserved			

Notes to [Table 2-66](#) (page 120):

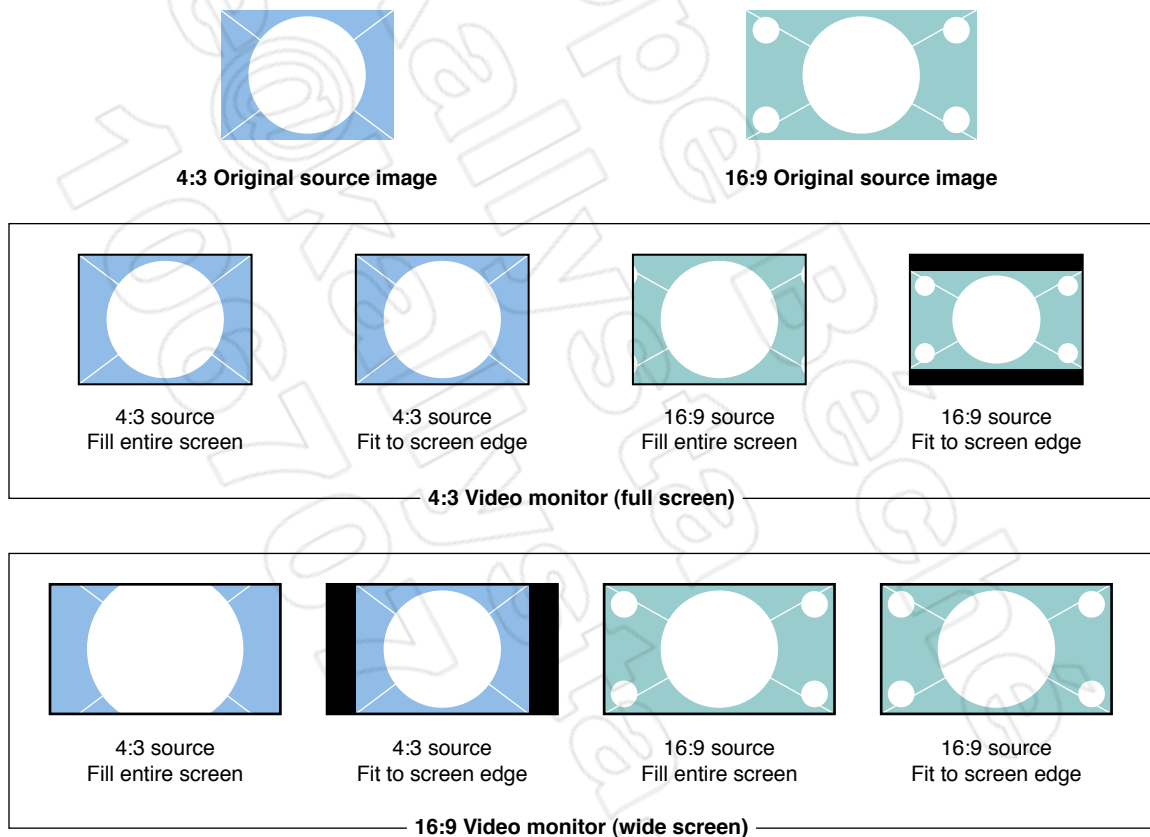
1. If the source media has a 4:3 (fullscreen) aspect ratio and the destination monitor has a 16:9 (widescreen) aspect ratio, the video is enlarged to eliminate the pillarbox bars normally present on the left and right of the screen. This may result in the top and bottom edges of the video being enlarged beyond the limits of the screen and cut off. If the source media has a 16:9 (widescreen) aspect ratio and the destination monitor has a 4:3 (fullscreen) aspect ratio, the video is enlarged to eliminate the letterbox bars normally present on the top and bottom of the screen. This may result in the left and right edges of the video being enlarged beyond the limits of the screen and cut off. See [Figure 2-1](#) (page 124).
2. If the source media has a 4:3 (fullscreen) aspect ratio and the destination monitor has a 16:9 (widescreen) aspect ratio, the video is enlarged so that the image fills the screen vertically, but it may have pillarbox black bars on the left and right edges of the screen. If the source media has a 16:9 (widescreen) aspect ratio and the destination monitor has a 4:3 (fullscreen) aspect ratio, the video is enlarged so that the image fills the screen horizontally, but it may have letterbox black bars on the top and bottom edges of the screen. See [Figure 2-1](#) (page 124).

3. The line out usage preference may be used to route Apple device audio to either USB digital audio output or analog line out, only if the following are all true:
 - `GetiPodOptionsForLingo` shows that the Apple device supports the General lingo analog/USB audio routing bit
 - Accessory identification includes the Digital Audio lingo
 - The accessory sets the analog/digital audio output routing capabilities bit (bit 21) in its FID `AccCapsToken`.

The Apple device ignores the `bRestoreOnExit` parameter for this preference, and this parameter should be set to 1. The Apple device will be unconditionally restore it when the accessory is detached.

4. On the iPod touch, subtitle display is not a global setting; it is set for each content item.

Figure 2-1 Screen configuration examples



Authentication Note: To receive video signals from the iPhone and from most Apple devices, and to set their preferences, an accessory must be authenticated; the only class ID that an accessory can get or set without being authenticated is 0x03 (line-out usage). For the best user experience, the Apple device should be configured to the accessory's line out and video out preferences as soon as possible. If an accessory has not set its preferences, the Apple device will make assumptions based on the accessory's Accessory Identify Resistor and declared lingo. These assumptions may not match the accessory's needs, so it is strongly recommended that accessories set their Apple device preferences promptly. The earliest time at which an accessory can set its video preferences is immediately after the Apple device sends it an `AckDevAuthenticationInfo` command with a value of 0x00 (success status). The 5G iPod is exempt from the authentication requirement.

Command 0x2A: RetiPodPreferences

Direction: Apple device to Accessory

The Apple device sends this command in response to a `GetiPodPreferences` command from the accessory. In byte 5 it echoes the requested preference class ID, and in byte 6 it sends the current preference setting for that class.

Table 2-67 RetiPodPreferences packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x00	Lingo ID: General lingo
4	0x2A	Command ID: RetiPodPreferences
5	0xNN	Preference class ID (see Table 2-66 (page 120))
6	0xNN	Preference setting ID (see Table 2-66 (page 120))
7	0xNN	Checksum

Command 0x2B: SetiPodPreferences

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to set a specific preference. It sends the preference class ID in byte 5 and the setting ID in byte 6. If byte 7, restore on exit, is set to 0x01, then the Apple device restores the original setting for this preference when the accessory is disconnected; if it is 0x00, it does not perform the restore. Other values of byte 7 are illegal.

Note: This command fails if the Apple device does not detect a valid R_{ID} resistor. See “Accessory Detect and Identify” in *MFi Accessory Hardware Specification*.

Although Apple device options need not be used to turn on video output from the Apple device, the Apple device has no default settings. It will reflect the options the user has entered in its settings menu, for which the accessory should not make any assumptions. The following are recommended option-setting practices for accessory designs:

- The accessory should set the video signal format (NTSC or PAL) to ensure signal compatibility with the attached display.
- The accessory should set the video monitor aspect ratio, if possible, to ensure image compatibility with the attached display.
- The accessory should select the audio output path (line out or digital audio out) to ensure that audio will be routed appropriately with its associated video.
- The accessory should always select the Restore on Exit flag to return the Apple device to its previous state when it is disconnected.

Note: Accessories that use the IDPS process must set their video preferences using `SetFIDTokenValues`. Accessories that identify using `IdentifyDeviceLingoes` must set their video preferences after the authentication process has begun and after the accessory has received an `AckDevAuthenticationInfo` command with a value of 0x00. Attempts to set video preferences earlier will result in an ACK command with an error status. See “Authentication Note” (page 125).

Table 2-68 SetiPodPreferences packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet
3	0x00	Lingo ID: General lingo
4	0x2B	Command ID: SetiPodPreferences
5	0xNN	Preference class ID (see Table 2-66 (page 120))
6	0xNN	Preference setting ID (see Table 2-66 (page 120))
7	0xNN	Restore on exit
8	0xNN	Checksum

Note: The Apple device's line-out state is always restored, regardless of the Restore-on-Exit setting. On some Apple devices, the video-out state is also always restored, regardless of the Restore-on-Exit setting.

Command 0x35: GetUIMode

Direction: Accessory to Apple device

The accessory sends this command to the attached Apple device to determine its current user interface mode. The Apple device replies by sending a RetUIMode command.

Table 2-69 GetUIMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x00	Lingo ID: General lingo
4	0x35	Command ID: GetUIMode
5	0xNN	Checksum

Command 0x36: RetUIMode

Direction: Apple device to Accessory

The Apple device sends this command in response to a GetUIMode command, to report its current user interface mode. The modes it may report are listed in [Table 2-71](#) (page 128).

Table 2-70 RetUIMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x00	Lingo ID: General lingo
4	0x36	Command ID: RetUIMode
5	0xNN	UIMode: The Apple device's current user interface mode; see Table 2-71 (page 128).
6	0xNN	Checksum

Table 2-71 User interface mode values

Value	Meaning
0x00	Standard Apple device operating mode
0x01	Extended Interface mode; see “The Extended Interface Protocol” in <i>iPod Accessory Protocol Interface Specification</i> .
0x02	iPod Out mode
0x03-0xFF	Reserved

Command 0x37: SetUIMode

Direction: Accessory to Apple device

The accessory sends this command to an Apple device to change its user interface mode. If the Apple device is successful in switching UI modes or is currently in the mode that the `SetUIMode` command requests, it returns a General lingo ACK command with success status. Otherwise, it returns an ACK command with the appropriate failure status.

Table 2-72 SetUIMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x00	Lingo ID: General lingo
4	0x37	Command ID: SetUIMode
5	0xNN	UIMode: The user interface mode to be set; see Table 2-71 (page 128).
6	0xNN	Checksum

Note: The General Lingo commands previously used to change user interface modes (`RequestRemoteUIMode` (0x03), `ReturnRemoteUIMode` (0x04), `EnterRemoteUIMode` (0x05), and `ExitRemoteUIMode` (0x06)) still work on Apple devices that support iPod Out. Extended Interface mode is available only if the accessory identifies itself successfully for the Extended Interface lingo (Lingo 0x04). Either `SetUIMode` with UIMode set to 0x01 or `EnterRemoteUIMode` places the Apple device in Extended Interface mode; either `SetUIMode` with UIMode set to 0x00 or `ExitRemoteUIMode` places the Apple device in its Standard operating mode.

Command 0x38: StartIDPS

Direction: Accessory to Apple device

This command is sent by the accessory to start the Identify Device Preferences and Settings (IDPS) process. When the Apple device sends a General lingo ACK command with a status of Success (0x00) in response to `StartIDPS`, it enters the IDPS process described in "Using IDPS" (page 519). If the accessory sends `StartIDPS` again, while the Apple device is in the IDPS process, the Apple device will restart the IDPS process.

If the Apple device sends a General lingo ACK command with a status of Maximum Connections (0x15) the accessory must not continue with IDPS or any other identification process, because the Apple device has already reached its maximum allowed number of accessory connections.

The accessory must start and complete the IDPS process (defined as sending `EndIDPS` successfully) within a total time of 3 seconds after the rising edge of power from the Apple device on pin 13 (Accessory Power) of the 30-pin connector. The accessory may continue the IDPS process only after it has received an ACK response to `StartIDPS` whose transaction ID matches that of the accessory's most recent `StartIDPS` command. Each `StartIDPS` command renders previous `StartIDPS` commands invalid, even if they are subsequently acknowledged.

After the IDPS process has finished, the Apple device sends the accessory an `IDPSStatus` command. If this command indicates an unsuccessful IDPS process, the accessory can retry `StartIDPS` within the 3-second total time limit. If the IDPS process times out, the accessory won't be able to retry IDPS unless it is detached and re-attached, or the Apple device sends a `RequestIdentify` command.

If the accessory is in the IDPS process and cannot complete it successfully, it must send `EndIDPS` with a status value of 0x02 before either retrying IDPS or sending `IdentifyDeviceLingoes`. If the accessory does this within 3 seconds of sending `StartIDPS`, the Apple device will send an `IDPSStatus` of 0x04 and will let the accessory retry the IDPS process or send `IdentifyDeviceLingoes`. If the accessory fails to do this within 3 seconds of sending `StartIDPS`, the Apple device will not let the accessory retry IDPS and will send an `IDPSStatus` of 0x05 in response to the accessory's `EndIDPS` command. After that, the Apple device will accept only `IdentifyDeviceLingoes`.

If an accessory has already completed IDPS successfully, inadvertently sending a `StartIDPS` or `IdentifyDeviceLingoes` command resets its authentication and identification states. The accessory must repeat the IDPS and authentication processes.

When an Apple device transitions from sleep to power on, it sends a `RequestIdentify` command to the accessory. The accessory must respond with `StartIDPS` and go through the IDPS process. After an Apple device transitions from hibernation and the accessory detects current from the Apple device on the Accessory Power line of the 30-pin connector, the accessory must wait at least 80 ms, send `StartIDPS`, and complete the IDPS process.

Table 2-73 `StartIDPS` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x38	Command ID: <code>StartIDPS</code>
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).

Byte number	Value	Comment
6	0xNN	transID [bits 7:0]
7	0xNN	Checksum

Command 0x39: SetFIDTokenValues

Direction: Accessory to Apple device

During the IDPS process, the accessory uses this command to send the Apple device a full ID string (FID). This string contains token-value fields that the Apple device is able to parse during the accessory identification process. The Apple device accepts this command only if the accessory previously initiated the IDPS process by sending a `StartIDPS` command.

The accessory may send this command multiple times with different transaction IDs, but it must put as many token-value fields as possible into each command. No command may exceed a value of 500 bytes in its length-of-packet-payload field. The accessory must wait for the Apple device to respond to each `SetFIDTokenValues` command with a `RetFIDTokenValueACKs` command before sending the next. If the same token is sent multiple times, the last value will be stored. If a `SetFIDTokenValues` packet is malformed and the Apple device cannot parse one or more token-value fields, the Apple device will respond by sending a General lingo ACK command with a nonzero status.

Note: The accessory must send certain token-value fields to the Apple device before the accessory can send `EndIDPS` to complete the IDPS process and proceed with authentication. See [Table 2-76](#) (page 131) for these requirements.

Table 2-74 SetFIDTokenValues packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x39	Command ID: SetFIDTokenValues
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	numFIDTokenValues: The number of token-value fields the accessory is setting by this command. If the number of token-value fields the Apple device parses from the command doesn't match this value, the Apple device returns a nonzero ACK and accepts no token-value fields.

Byte number	Value	Comment
8- <i>NN</i>	0x <i>NN</i>	FIDTokenValues : Token-value fields containing information the accessory sends to the Apple device before going through authentication. See Table 2-75 (page 131) for the format of each field.
(last byte)	0x <i>NN</i>	Checksum

Table 2-75 FIDTokenValues field format

Byte	Size	Name	Description
0	0x01	length	Length of this token-value field in bytes, not including this byte.
1	0x01	FIDType	First byte of token ID; see Table 2-76 (page 131).
2	0x01	FIDSubtype	Second byte of token ID; see Table 2-76 (page 131).
3- <i>NN</i>	0x <i>NN</i>	data	Data corresponding to token; see Table 2-77 (page 132) through Table 2-89 (page 138). All multibyte elements in the data field must be big-endian.

Table 2-76 FIDTokenValues tokens

FIDType	FIDSubtype	Token name	Requirements	Format
0	0	IdentifyToken	Required; must be the first token sent.	See Table 2-77 (page 132).
0	1	AccCapsToken	Required	See Table 2-79 (page 133).
0	2	AccInfoToken	Accessory Info Types 0x01, 0x04, 0x05, 0x06, 0x07, and 0x0C required; other types optional. See Table 2-82 (page 135).	See Table 2-81 (page 135).
0	3	iPodPreferenceToken	Recommended	See Table 2-84 (page 136).
0	4	EAPProtocolToken	Required for accessories that communicate with applications.	See Table 2-85 (page 136).
0	5	BundleSeedIDPref-Token	Required for accessories that communicate with applications.	See Table 2-86 (page 137).
0	7	ScreenInfoToken	Required if the accessory supports iPod Out mode; see "Lingo 0x0D: iPod Out Lingo" (page 374).	See Table 2-89 (page 138).

FIDType	FIDSubtype	Token name	Requirements	Format
0	8	EAProtocolMetadataToken	Optional; lets accessories declare their preference for matching with applications on the app store.	See Table 2-87 (page 137).
1	0	MicrophoneCapsToken	Deprecated; do not use in new designs.	

WARNING: The IDPS process in some older Apple devices can handle only `IdentifyToken`, `AccCapsToken`, `AccInfoToken`, `iPodPreferenceToken`, `EAProtocolToken`, and `BundleSeedIDPrefToken` tokens. The accessory can determine this by checking for a value of 0 in bit 23 in the General lingo `RetiPodOptionsForLingo` option bits; see [Table 2-138](#) (page 167).

The IDPS process will fail unless the Apple device successfully receives all the tokens listed as required in [Table 2-76](#) (page 131). The `IdentifyToken` token must be sent at the beginning of the first `SetFIDTokenValues` packet. For the best user experience, the accessory should also return all the recommended tokens that it supports.

The accessory will also fail the IDPS process if it tries to set a preference in the `iPodPreferenceToken` that requires an accessory capability not previously declared in the `AccCapsToken`. An example of such a failure would be to declare no line-out support in the `AccCapsToken` but then try to set a line-out usage preference in the `iPodPreferenceToken`.

Although an `iPodPreferenceToken` is not required, Apple devices do not have default settings and the accessory should make no assumptions about their preferences. During the IDPS process, the accessory should set every preference it will require. The accessory can later change Apple device preferences by using the General lingo `SetiPodPreferences` command.

Table 2-77 `IdentifyToken` format

Name	Bytes	Value	Description
length	1	0xNN	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x00	Second byte of token ID.
numLingoes	1	0xNN	Number of bytes in <code>accessoryLingoes</code> .
accessoryLingoes	NN	<var>	One byte for each lingo the accessory supports. For example, 0x00, 0x02, and 0x04 = General lingo + Simple Remote lingo + Extended Interface lingo, with <code>numLingoes</code> set to 0x03 and <code>length</code> set to 0x0E. The General lingo (0x00) must always be included.
DeviceOptions	4	0xNNNNNNNN	Accessory's device options; see Table 2-78 (page 133). The accessory must request immediate Authentication 2.0.

Name	Bytes	Value	Description
DeviceID	4	0xNNNNNNNN	Accessory's unique identifier, supplied by its iPod Authentication Coprocessor.

Table 2-78 DeviceOptions bits in IdentifyToken

Bits	Meaning
1:0	Authentication control bits. These bits have the following meanings: 00–01 = reserved 10 = authenticate immediately after identification (required for Authentication 2.0). 11 = reserved
3:2	Power control bits. These bits have the following meanings: 00 = low power mode, as defined in the section “Accessory Power Policy” in <i>MFi Accessory Hardware Specification</i> 01 = intermittent high power; the accessory requires more than low power from the Apple device (up to 100mA maximum) during playback operation 10 = reserved 11 = constant high power; the accessory requires more than low power from the Apple device (up to 100 mA maximum) at all times. This mode must not be declared unless the accessory provides power as specified in “Supplying USB Power” in <i>MFi Accessory Hardware Specification</i> .
31:4	Reserved; set to 0

Table 2-79 AccCapsToken format

Name	Bytes	Value	Description
length	1	0x0A	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x01	Second byte of token ID.
accCapsBitmask	8	0xNNNNNNNNNNNNNNNN	Accessory capabilities; see Table 2-80 (page 133).

Table 2-80 Accessory capabilities bit values

Bit	Accessory capability
00	Analog line-out (accessory consumes Apple device line-out)
01	Analog line-in (accessory supplies line-in to the Apple device)
02	Analog video-out (accessory consumes Apple device video-out)

Bit	Accessory capability
03	Reserved (set to 0)
04	USB audio-out (accessory consumes digital audio from Apple device in USB Device mode)
05–08	Reserved (set to 0)
09	Accessory supports communication with an application.
10	Reserved (set to 0)
11	Accessory checks Apple device volume, either by registering for Display Remote lingo notifications or by sending <code>GetiPodStateInfo</code> commands with an <code>infoType</code> of 0x04 or 0x10.
12–16	Reserved (set to 0)
17	Accessory uses Apple device user interface accessibility features; see "User Interface Accessibility Commands" (page 198).
18	The accessory can safely handle asynchronous playback state changes from the Apple device, even when the accessory did not initiate those changes; see "Playback Status Notifications" (page 418). Such accessories may operate alongside other accessories connected to a single Apple device. All Bluetooth accessories must support this capability.
19	Accessory can handle a multi-packet response from the Apple device, even when it is mixed with other packets that may include notifications and responses to other commands. This bit must be set for an accessory to cancel long data downloads; see "Command 0x50: CancelCommand" (page 178).
20	Reserved (set to 0)
21	<p>1 = To maintain digital audio/video synchronization when playing video media, the USB host accessory will switch between USB digital audio and analog line out audio routings when playing video and non-video media. This bit may be set only if these conditions are true:</p> <ul style="list-style-type: none"> ■ The accessory is a USB host and the Apple device is a USB device, and the accessory identifies itself for the Digital Audio lingo. ■ The Digital Audio lingo does not support the <code>SetVideoDelay</code> command; see <code>GetiPodOptionsForLingo</code> for the Digital Audio lingo. ■ The General lingo supports the analog/USB audio routing feature using the <code>SetiPodPreferences</code> line out preference; see <code>GetiPodOptionsForLingo</code> for the General lingo. <p>0 = The accessory as a USB host will not switch audio routings when playing video and non-video media using the USB digital audio routing. If the accessory has identified itself for the Digital Audio lingo, sending a <code>SetiPodPreferences</code> command with the line out class ID has no effect.</p>
22–63	Reserved (set to 0)

Table 2-81 AccInfoToken format

Name	Bytes	Value	Description
length	1	0xNN	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x02	Second byte of token ID.
accInfoType	1	0xNN	Accessory info type; see Table 2-82 (page 135).
accInfo	NN	<var>	Accessory info; see Table 2-82 (page 135).

Table 2-82 Accessory Info Type values

Value	Meaning	Parameter
0x00	Reserved	
0x01	Accessory name	Null-terminated UTF-8 string (64 bytes maximum, including terminator)
0x02	Reserved	
0x03	Reserved	
0x04	Accessory firmware version	3 bytes
0x05	Accessory hardware version	3 bytes
0x06	Accessory manufacturer	Null-terminated UTF-8 string (64 bytes maximum, including terminator)
0x07	Accessory model number	Null-terminated UTF-8 string (64 bytes maximum, including terminator)
0x08	Accessory serial number	Null-terminated UTF-8 string (64 bytes maximum, including terminator)
0x09	Accessory incoming maximum packet size	2 bytes. The maximum packet size must be bigger or equal to 128 bytes and smaller or equal to 65535 (0xFFFF) bytes.
0x0A	Reserved	
0x0B	Accessory status	32-bit mask; see Table 2-64 (page 120).
0x0C	Accessory RF certifications	32-bit <code>RF Certification Declaration</code> (RF certification mask); see Table 2-83 (page 136). Each bit may be set only if the accessory has been tested by an Apple-authorized independent laboratory and has passed all the RF tests required for that class. Otherwise all bits must be 0. See Apple's <i>MFi Accessory Testing Specification</i> for details of the tests required.

Value	Meaning	Parameter
0x0D–0xFF	Reserved	

Table 2-83 Accessory RF certification declarations

Mask bit	Certification	Products
00	Class 1	iPhone, iPhone 3G, iPhone 3GS
01	Class 2	iPhone 4 (GSM model)
02	Class 3	Reserved
03	Class 4	iPhone 4 (CDMA model)
31:04	Reserved	

Table 2-84 iPodPreferenceToken format

Name	Bytes	Value	Description
length	1	0x05	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x03	Second byte of token ID.
iPodPrefClass	1	0xNN	Apple device preference class ID; see Table 2-66 (page 120).
prefClassSetting	1	0xNN	Apple device preference setting ID; see Table 2-66 (page 120).
restoreOnExit	1	0x01	Must be set to 0x01.

Table 2-85 EAPProtocolToken format

Name	Bytes	Value	Description
length	1	0xNN	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x04	Second byte of token ID.
protocolIndex	1	0xNN	A unique protocol index, starting from 1. Subsequent indexes in other EAPProtocolToken token-value fields must increment by 1.
protocolString	NN	<var>	A null-terminated UTF-8 reverse-DNS string (e.g. "com.acme.gadget"). This string will be compared (without considering case) to strings presented by applications on the iPhone or iPod touch.

Note: If an accessory sends a duplicate `protocolString` or repeats a `protocolIndex` number, the last received `EAPProtocolToken` will be used; any information from a previous `EAPProtocolToken` will be overwritten.

Table 2-86 BundleSeedIDPrefToken format

Name	Bytes	Value	Description
length	1	0x0D	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x05	Second byte of token ID.
BundleSeedIDString	11	<var>	A null-terminated UTF-8 string (e.g. "A1B2C3D4E5") that identifies the vendor of a preferred application, with case sensitivity. This string is derived from the vendor's App ID assigned by Apple.

Table 2-87 EAPProtocolMetadataToken format

Name	Bytes	Value	Description
length	1	0xNN	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x08	Second byte of token ID.
protocolIndex	1	0xNN	A unique protocol index, starting from 1. Subsequent indexes in other <code>EAPProtocolToken</code> token-value fields must increment by 1.
metadataType	1	NN	See Table 2-88 (page 137).
metadataInfo	NN	<var>	Reserved; leave this field empty.

Table 2-88 MetadataType values

Value	Meaning
0x00	The Apple device will hide as much information about matching applications as possible. The Apple device will not match the accessory's <code>BundleSeedID</code> or protocols with any application on the app store. The Apple device will try to omit the "Go to the App Store" notification and suppress "Find App For This Accessory" in the Settings->About screen.
0x01	The Apple device will match this <code>protocolIndex</code> with the <code>BundleSeedID</code> and with applications on the app store if there is no matching application present on the Apple device. The accessory must accept the default behavior of the Apple device's firmware with regards to matching protocols and displaying app store notifications. For example, if the Apple device's default behavior is that a "Go to the App Store" notification is presented at most 3 times if there is no matching application present on the Apple device, the accessory must accept that behavior.

Value	Meaning
0x02	The Apple device will always match this <code>protocolIndex</code> with the <code>BundleSeedID</code> and applications on the app store if there is no matching app present on the Apple device. This overrides any Apple device default behavior. For example, if the Apple device's default behavior is that a "Go to the App Store" notification is presented at most 3 times if there is no matching app present on the Apple device, the accessory overrides this behavior and forces the notification to be displayed every time.
0x03	Do not match this <code>protocolIndex</code> with the <code>BundleSeedID</code> and applications on the app store, regardless of whether there is a matching app present on the Apple device or not. This will prevent the possibility of the app store notification being displayed to the user. Although the "Go to the App Store" notification will be suppressed, the "Find App For This Accessory" button will continue to show in the Settings->About screen on the Apple device.
0x04-0xFF	Reserved.

Table 2-89 ScreenInfoToken format

Name	Bytes	Value	Description
length	1	0x10	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x07	Second byte of token ID.
totalScreenWidthInches	2	0xNN	Accessory's approximate screen width in inches
totalScreenHeightInches	2	0xNN	Accessory's approximate screen height in inches
totalScreenWidthPixels	2	0xNN	Number of pixels along the accessory's screen width
totalScreenHeightPixels	2	0xNN	Number of pixels along the accessory's screen height
iPodOutScreenWidthPixels	2	0xNN	Number of pixels along the iPod Out screen width allotment
iPodOutScreen-HeightPixels	2	0xNN	Number of pixels along the iPod Out screen height allotment
screenFeaturesMask	1	0xNN	A bitmask with bits set to represent the accessory screen's features; see Table 2-90 (page 139).
screenGammaValue	1	0xNN	A representation of the accessory display's gamma value. The Apple device obtains the gamma value from this byte by dividing it by 100 and adding 1.00. For example, a gamma of 2.200 is represented by a <code>screenGammaValue</code> of 120.

Table 2-90 Accessory screen features

Bit	Meaning
00	The accessory has a color display
07-01	Reserved

Command 0x3A: RetFIDTokenValueACKs

Direction: Apple device to Accessory

The Apple device sends this command in response to each `SetFIDTokenValues` packet from the accessory. The number of token-value fields contained in `FIDTokenValueACKs` matches exactly the number of fields the accessory sent in its `SetFIDTokenValues` command. The different statuses returned to the accessory let it determine whether to retry a failed token-value field.

Table 2-91 RetFIDTokenValueACKs packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x3A	Command ID: RetFIDTokenValueACKs
5	0xNN	transID [bits 15:8]: Transaction ID of the <code>SetFIDTokenValues</code> packet being responded to; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	numFIDTokenValueACKs: The number of token-value fields the Apple device is acknowledging. This number matches the number the accessory sent in the <code>SetFIDTokenValues</code> command.
8-NN	0xNN	FIDTokenValueACKs: Acknowledgement values for the token-value fields sent by <code>SetFIDTokenValues</code> . See Table 2-92 (page 139) through Table 2-101 (page 142).
(last byte)	0xNN	Checksum

Table 2-92 Acknowledgment format for IdentifyToken

Name	Bytes	Value	Description
length	1	0x03	Length of this token-value field in bytes, not including this byte.

Name	Bytes	Value	Description
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x00	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see Table 2-93 (page 140).
lingoIDs	0xNN	0xNN	If ACKStatus = 4, IDs of busy lingo; see Table 2-93 (page 140).

Table 2-93 Acknowledgment status codes

Value	Description
0	Token-value field accepted.
1	Required token-value field failed.
2	Optional token-value field recognized, but failed; the accessory may retry, either with another <code>SetFIDTokenValues</code> command while in the IDPS process or with another iAP command that emulates the desired setting. The accessory should not allow this status return to prevent it from completing IDPS process.
3	Token not supported; the accessory must not retry with this Apple device.
4	Lingoes busy. This status responds to an <code>IdentifyToken</code> that tries to register for lingoes that are busy on another port. The ID numbers of the busy lingoes are appended to the token. See <code>lingoIDs</code> in Table 2-92 (page 139).
5	Maximum connections reached. This status responds to an <code>IdentifyToken</code> if another accessory is connected and the calling accessory cannot connect to the Apple device.
6–255	Reserved

Table 2-94 Acknowledgment format for `AccCapsToken`

Name	Bytes	Value	Description
length	1	0x03	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x01	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see Table 2-93 (page 140).

Table 2-95 Acknowledgment format for `AccInfoToken`

Name	Bytes	Value	Description
length	1	0x04	Length of this token-value field in bytes, not including this byte.

Name	Bytes	Value	Description
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x02	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see Table 2-93 (page 140).
accInfoType	1	0xNN	Accessory info type in token being acknowledged.

Table 2-96 Acknowledgment format for iPodPreferenceToken

Name	Bytes	Value	Description
length	1	0x04	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x03	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see Table 2-93 (page 140).
iPodPrefClass	1	0xNN	Apple device preference class in token being acknowledged.

Table 2-97 Acknowledgment format for EAPProtocolToken

Name	Bytes	Value	Description
length	1	0x04	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x04	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see Table 2-93 (page 140).
protocolIndex	1	0xNN	Protocol index in token being acknowledged.

Table 2-98 Acknowledgment format for BundleSeedIDPrefToken

Name	Bytes	Value	Description
length	1	0x03	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x05	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see Table 2-93 (page 140).

Table 2-99 Acknowledgment format for `EAPProtocolMetadataToken`

Name	Bytes	Value	Description
length	1	0x03	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x08	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see Table 2-93 (page 140).

Table 2-100 Acknowledgment format for `MicCapsToken`

Name	Bytes	Value	Description
length	1	0x03	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x01	First byte of token ID.
FIDSubtype	1	0x00	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see Table 2-93 (page 140).

Table 2-101 Acknowledgment format for `ScreenInfoToken`

Name	Bytes	Value	Description
length	1	0x03	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x00	First byte of token ID.
FIDSubtype	1	0x07	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see Table 2-102 (page 142).

Table 2-102 Acknowledgment status codes

Value	Description
0	Token-value field accepted.
1	Required token-value field failed.
2	Optional token-value field recognized, but failed; the accessory may retry, either with another <code>SetFIDTokenValues</code> command while in the IDPS process or with another iAP command that emulates the desired setting. The accessory should not allow this status return to prevent it from completing IDPS process.
3	Token not supported; the accessory must not retry with this Apple device.
4	Lingoes busy. This status responds to an <code>IdentifyToken</code> that tries to register for lingoes that are busy on another port. The ID numbers of the busy lingoes are appended to the token.

Value	Description
5	Maximum connections reached. This status responds to an <code>IdentifyToken</code> if another accessory is connected and the calling accessory cannot connect to the Apple device.
6–255	Reserved

Command 0x3B: EndIDPS

Direction: Accessory to Apple device

This command is sent by the accessory to end the IDPS process. The Apple device takes different actions depending on the value of `accIDPSStatus`.

If the accessory sends this command while the Apple device is not in the IDPS process, the Apple device responds with an `ACK` command that passes a nonzero status. If the Apple device is in the IDPS process and the accessory sends this command with an unsupported `accEndIDPSStatus` value, the Apple device remains in the IDPS process.

Table 2-103 EndIDPS packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x3B	Command ID: EndIDPS
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>accEndIDPSStatus</code> : The accessory's IDPS status, which determines what action the Apple device takes after receiving EndIDPS. See Table 2-104 (page 143).
8	0xNN	Checksum

Table 2-104 `accEndIDPSStatus` values

Value	Actions to be taken
0	The accessory has finished with IDPS, and asks the Apple device to continue with authentication if the Apple device sends a <code>IDPSStatus</code> command with success status.
1	The accessory asks to reset all IDPS information it has sent to the Apple device. The accessory must then send another <code>StartIDPS</code> command and successfully complete the IDPS process within the time limit specified in "Command 0x38: StartIDPS" (page 128).

Value	Actions to be taken
2	The accessory has determined that the Apple device doesn't support features the accessory needs. The accessory is abandoning the IDPS process.
3	The accessory has finished with IDPS on the current transport link and is restarting IDPS on another transport link. The accessory must cease traffic on the old link immediately and restart IDPS on the new link within 2 seconds.
4–255	Reserved

Note: In the future, an accessory may be connected to an Apple device that supports IDPS but doesn't recognize a specific token-value field. To handle this case, the accessory must respond to the `accIDPSStatus` value of 2. For example, if the accessory needs to set a token-value field that the attached Apple device cannot parse, it may determine that it can't work with the Apple device and hence must discontinue the identification process.

Command 0x3C: IDPSStatus

Direction: Apple device to Accessory

This command is sent by the Apple device after an accessory sends `EndIDPS`. The Apple device determines whether all required token-value fields were successfully sent by the accessory, and sends a `status` value that indicates whether authentication will proceed and the Apple device will apply the tokens, or the accessory failed to identify itself properly.

Note: After receiving an `EndIDPS` with status 0, the Apple device evaluates the set of FID tokens that the accessory sent for completeness and consistency. At this point the accessory will fail the IDPS process if it has set a preference in the `iPodPreferenceToken` that requires an accessory capability not declared in the `AccCapsToken`. An example of such an inconsistency would be to declare no line-out support in the `AccCapsToken` but also set a line-out usage preference in the `iPodPreferenceToken`.

Table 2-105 IDPSStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x3C	Command ID: IDPSStatus
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	<code>transID</code> [bits 7:0]

Byte number	Value	Comment
7	0xNN	status: A value that indicates whether the accessory successfully identified during the IDPS sequence. Combined with the value of <code>accEndIDPSStatus</code> sent by the accessory, it determines the Apple device's action. See Table 2-106 (page 145).
8	0xNN	Checksum

Table 2-106 Apple device actions in response to `accEndIDPSStatus` and IDPS status

accEndIDPSStatus	IDPS status	Action taken by the Apple device
0	0	The Apple device received all required token-value fields; authentication will proceed. Optional token-value fields may or may not have been received, but their status will not block authentication.
	1	One or more required token-value fields were rejected by an <code>FIDTokenValueACK</code> command and were not correctly resent; IDPS fails.
	2	One or more required token-value fields were missing; IDPS fails.
	3	One or more required token-value fields were rejected by an <code>FIDTokenValueACK</code> command and were not correctly resent, plus one or more required token-value fields were missing; IDPS fails.
1	4	The IDPS time limit was not exceeded; the accessory may retry IDPS or send <code>IdentifyDeviceLingoes</code> .
	5	The IDPS time limit was reached; the accessory cannot retry IDPS but may send <code>IdentifyDeviceLingoes</code> .
2	6	The Apple device will not accept any token-value fields and will not continue with authentication. The accessory may send <code>IdentifyDeviceLingoes</code> but not <code>StartIDPS</code> .
0	7	<p>IDPS fails for one or more of the following reasons:</p> <ul style="list-style-type: none"> ■ The accessory sent an Apple device Preference Token for preference class 0x03 (line-out) without indicating line-out support in the <code>accCapsBitmask</code> field of its Accessory Capabilities Token. ■ The accessory sent an Apple device Preference Token for at least one of preference classes 0x00 (video-out), 0x01 (video-out screen configuration), 0x02 (video-out signal format), 0x08 (video-out connection), or 0x0A (video monitor aspect ratio) without indicating video-out support in the <code>accCapsBitmask</code> field of its Accessory Capabilities Token. ■ The accessory set any EA Protocol Tokens, or a Preferred Bundle Seed ID, without setting bit 09 in its <code>AccCapsToken</code>. ■ The accessory identified for the Digital Audio Lingo in its Identify token without setting the USB Audio bit (bit 04) in its <code>AccCapsToken</code>.

accEndIDPSStatus	IDPS status	Action taken by the Apple device
NN	8–255	Reserved

Note: When a failed IDPS process is retried, the Apple device does not retain any successfully set token-value fields from the failed IDPS process. Accessories must send all token-value fields again.

Command 0x3F: OpenDataSessionForProtocol

Direction: Apple device to Accessory

The Apple device sends this command to tell the accessory that a data stream has been opened between the accessory and an iOS application. It sends the accessory the `sessionID` of the data stream and the `protocolIndex` to which the open session corresponds. All communications between the accessory and the application for the given `sessionID` are assumed to use the protocol specified by `protocolIndex`.

The accessory must send a `DevACK` command in response. A success status means the accessory can support a new open data stream. If the accessory sends a `DevACK` command with other than success status, the Apple device informs the application that the accessory has refused the data stream connection.

Table 2-107 OpenDataSessionForProtocol packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x3F	Command ID: OpenDataSessionForProtocol
5	0xNN	transID [bits 15:8]: Transaction ID accessory must use when acknowledging this command; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	sessionID [bits 15:8]: Session ID for subsequent data transfer commands
8	0xNN	sessionID [bits 7:0]
9	0xNN	protocolIndex: Index of the protocol for which this session is being opened; see Table 2-85 (page 136).
10	0xNN	Checksum

Command 0x40: CloseDataSession

Direction: Apple device to Accessory

The Apple device sends this command to tell the accessory that the data stream on the given `sessionID` is closing, ending communication. The accessory must send a `DevACK` response, but the data stream will remain closed regardless of the `DevACK` status. If the accessory sends any more `DevDataTransfer` packets for the closed session they will be acknowledged with an Unknown Category status (0x01). The Apple device never retries a `CloseDataSession` command.

Table 2-108 CloseDataSession packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x40	Command ID: CloseDataSession
5	0xNN	transID [bits 15:8]: Transaction ID accessory must use when acknowledging this command; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	sessionID [bits 15:8]: Session ID of the session being closed
8	0xNN	sessionID [bits 7:0]
9	0xNN	Checksum

Command 0x41: DevACK

Direction: Accessory to Apple device

The accessory must use this command to acknowledge General lingo commands 0x3F (`OpenDataSessionForProtocol`), 0x40 (`CloseDataSession`), and 0x43 (`iPodDataTransfer`) from the Apple device. This command must not be used for any other purpose.

Table 2-109 DevACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload

Byte number	Value	Comment
3	0x00	Lingo ID: General lingo
4	0x41	Command ID: DevACK
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	ackStatus : Status of command being acknowledged: must be either 0x00 (Success) or 0x04 (Bad Parameter).
8	0xNN	cmdID : ID of command being acknowledged; must be 0x3F, 0x40, or 0x43.
9	0xNN	Checksum

Command 0x42: DevDataTransfer

Direction: Accessory to Apple device

The accessory uses this command to send data to an application listening for a specific `sessionID`, as established by a prior `OpenDataSessionForProtocol` command. The Apple device responds with a General lingo ACK command indicating whether or not the application was able to receive the data from this command.

If the accessory sends a `DevDataTransfer` command for a session that is closed or doesn't exist, the Apple device acknowledges it with an `Unknown Category` status (0x01). If the accessory sends a `DevDataTransfer` command for a session that the Apple device has opened but which the accessory has not acknowledged successfully, the Apple device acknowledges it with a `Command Failed` status (0x02).

During a communication session, the following rules apply to `DevDataTransfer` commands:

- The accessory should use `DevDataTransfer` commands whenever it has data it wants to stream to an application with an open session.
- Before sending the first `DevDataTransfer` command, the accessory must send a `RequestTransportMaxPayloadSize` command to determine the maximum payload size that it can stream.
- The accessory must not send a new `DevDataTransfer` packet until the previous one has been acknowledged (as successful or not) or a 500 ms timeout has expired.
- Upon receiving a successful acknowledgment of a `DevDataTransfer` command, the accessory may immediately send a new command containing available data for any session ID.
- If a 500 ms timeout occurs before a `DevDataTransfer` command is acknowledged successfully, the accessory may resend the same packet with the same transaction ID. If the accessory retries 10 times without receiving a successful acknowledgement, it must close the session and assume that the transfer failed.
- If a `DevDataTransfer` command is acknowledged with a `SessionWriteFailure` status (0x17), the accessory may send a packet with the same transaction ID containing the remaining data. If no additional bytes are successfully accepted after 10 such attempts, the accessory must close the session and assume that the transfer failed. See [Table 2-17](#) (page 87).

- If the accessory receives an `iPodNotification` command for flow control, it must not send any packets (including retries of `DevDataTransfer` packets) during the specified waiting time.

The accessory can choose either small or large packet format, as defined in "Command Packet Formats" (page 76). It can determine the maximum payload size that can be used with large packet format by sending a `RequestTransportMaxPayloadSize` command.

The small and large `DevDataTransfer` formats are shown in Table 2-110 (page 149) and Table 2-111 (page 149).

Table 2-110 `DevDataTransfer` small packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x42	Command ID: <code>DevDataTransfer</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>sessionID</code> [bits 15:8]: Session ID of the connection between the application and the accessory
8	0xNN	<code>sessionID</code> [bits 7:0]
9-NN	0xNN	<code>data</code> : Data the accessory sends to the listening application.
(last byte)	0xNN	Checksum

Table 2-111 `DevDataTransfer` large packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x00	Packet payload length marker
3	0xNN	Length of packet payload [bits 15:8]
4	0xNN	Length of packet payload [bits 7:0]
5	0x00	Lingo ID: General lingo
6	0x42	Command ID: <code>DevDataTransfer</code>

Byte number	Value	Comment
7	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
8	0xNN	transID [bits 7:0]
9	0xNN	sessionID [bits 15:8]: Session ID of the connection between the application and the accessory
10	0xNN	sessionID [bits 7:0]
11-NN	0xNN	data: Data the accessory sends to the listening application.
(last byte)	0xNN	Checksum

Command 0x43: iPodDataTransfer

Direction: Apple device to Accessory

The Apple device uses this command to send data to an accessory listening for a specific `sessionID`, as established by a prior `OpenDataSessionForProtocol` command. The accessory must respond with a `DevACK` command that passes a command ID of 0x43 and the same transaction ID as contained in the `iPodDataTransfer` packet.

Note: The Apple device will not send this command until accessory authentication has finished. The Authentication 2.0B process may last up to 2 seconds after accessory identification.

By default, the maximum `iPodDataTransfer` packet payload length is 1,018 bytes. The accessory can specify a different maximum length by sending an `AccInfoToken` with an `accInfoType` of 0x09. The length that can be set ranges from 128 to 65,535 bytes. Specifying less than 128 bytes will revert to 128. The packet payload length is the combined length of the lingo ID, command ID, transaction ID, session ID, and command data; the payload does not include the sync byte, packet start byte, packet payload length bytes, or packet payload checksum byte.

The Apple device may send data in either small packet format or large packet format, as defined in "Command Packet Formats" (page 76). The two `iPodDataTransfer` formats are shown in Table 2-112 (page 151) and Table 2-113 (page 151). Accessories must be prepared to receive either format unless they have specified a maximum packet payload length within the limits of the small packet format.

During a communication session, the following rules apply to `iPodDataTransfer` commands:

- The Apple device will send an initial `iPodDataTransfer` packet when an application has placed data into the accessory's input queue.
- Upon receiving an acknowledgment of an `iPodDataTransfer` command, the Apple device may immediately send a new command containing available data for any session ID.
- The accessory's acknowledgment of an `iPodDataTransfer` command with Success status (0x00) must signify that the packet has been received, it has been moved out of the accessory's lowest-level receive queue, and that the accessory is ready to receive another packet.

- The only circumstance under which an accessory may acknowledge an `iPodDataTransfer` command with a nonzero status is if no prior `OpenDataSessionForProtocol` command has established the command's session ID. In this case it must send a Bad Parameter (0x04) status, and the Apple device may close the session.
- If an `iPodDataTransfer` command is acknowledged with an error status, the Apple device may optionally purge its outbound packet queue of all pending `iPodDataTransfer` packets for that session ID. The Apple device will not otherwise discard any data in a session.
- If an accessory's data receive queue cannot accept another packet, then the accessory must not acknowledge the `iPodDataTransfer` command until packet acceptance is restored. This may cause the Apple device to retry the `iPodDataTransfer` command up to ten times. If the Apple device receives no acknowledgment after the tenth retry, it may close the session.
- If a 500 ms timeout occurs before an `iPodDataTransfer` command is acknowledged, the Apple device will resend the same packet with the same transaction ID.

Table 2-112 `iPodDataTransfer` small packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x43	Command ID: <code>iPodDataTransfer</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>sessionID</code> [bits 15:8]: Session ID of the connection between the application and the accessory
8	0xNN	<code>sessionID</code> [bits 7:0]
9-NN	0xNN	<code>data</code> : Data the application sends to the listening accessory.
(last byte)	0xNN	Checksum

Table 2-113 `iPodDataTransfer` large packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x00	Packet payload length marker
3	0xNN	Length of packet payload [bits 15:8]

Byte number	Value	Comment
4	0xNN	Length of packet payload [bits 7:0]
5	0x00	Lingo ID: General lingo
6	0x43	Command ID: iPodDataTransfer
7	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
8	0xNN	transID [bits 7:0]
9	0xNN	sessionID [bits 15:8]: Session ID of the connection between the application and the accessory
10	0xNN	sessionID [bits 7:0]
11-NN	0xNN	data : Data the application sends to the listening accessory.
(last byte)	0xNN	Checksum

Command 0x46: SetAccStatusNotification

Direction: Apple device to Accessory

The Apple device sends this command to the accessory to request status notifications for specific status types. The accessory must respond with a `RetAccStatusNotification` command, acknowledging the settings.

Immediately after sending a `RetAccStatusNotification` command, the accessory must send an initial `AccessoryStatusNotification` command for each notification for which the Apple device has registered, whether or not the corresponding status has changed.

The accessory must then send an `AccessoryStatusNotification` command for each status type set every time that status changes. Notifications should not be sent too frequently and notifications for the same status type must be sent no more often than once every 500 ms.

Table 2-114 SetAccStatusNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x46	Command ID: SetAccStatusNotification

Byte number	Value	Comment
5	0xNN	transID [bits 15:8]: Transaction ID of this command; see "Transaction IDs" (page 525). If the Apple device does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0x00	Status notifications mask (bits 31:24)
8	0x00	Status notifications mask (bits 23:16)
9	0x00	Status notifications mask (bits 15:8)
10	0xNN	Status notifications mask (bits 7:0); see Table 2-64 (page 120))
11	0xNN	Checksum

Currently the only accessory status value defined is the accessory fault condition (see Table 2-64 (page 120)). When the accessory sends an `AccessoryStatusNotification` command, it passes one of the fault type values shown in Table 2-119 (page 155) and one of the fault conditions shown in Table 2-120 (page 155).

Table 2-121 (page 155) lists a sample command flow that uses the General lingo 0x46–0x48 accessory status notification commands.

Command 0x47: RetAccStatusNotification

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to acknowledge receipt of a `SetAccStatusNotification` command. Bits in the status notification mask must be set for each notification type successfully enabled in the accessory.

Table 2-115 RetAccStatusNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x47	Command ID: RetAccStatusNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command; see "Transaction IDs" (page 525). If the Apple device does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]

Byte number	Value	Comment
7	0x00	Status notifications mask (bits 31:24)
8	0x00	Status notifications mask (bits 23:16)
9	0x00	Status notifications mask (bits 15:8)
10	0xNN	Status notifications mask (bits 7:0); see Table 2-64 (page 120))
11	0xNN	Checksum

[Table 2-121](#) (page 155) lists a sample command flow that uses the General lingo 0x46–0x48 accessory status notification commands.

Command 0x48: AccessoryStatusNotification

Direction: Accessory to Apple device

The accessory must send this command to the Apple device when there is a change in one of the status types for which the Apple device registered by previously sending a `SetAccStatusNotification` command. Notifications should not be sent too frequently and notifications for the same status type must be sent no more often than once every 500 ms.

Immediately after sending a `RetAccStatusNotification` command, the accessory must send an initial `AccessoryStatusNotification` command for each notification for which the Apple device registered, regardless of whether the corresponding status has changed.

Table 2-116 AccessoryStatusNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x48	Command ID: AccessoryStatusNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command; see " Transaction IDs " (page 525). If the Apple device does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0xNN	Status type; see Table 2-117 (page 155).
8–NN	0xNN	Status parameters; dependent on status type passed in byte 7.
NN	0xNN	Checksum

Table 2-117 Accessory status types

Value	Description	Parameters
0x00–0x01	Reserved	
0x02	Fault	2 bytes; see Table 2-118 (page 155)
0x03–0xFF	Reserved	

Table 2-118 Accessory status parameters (fault)

AccessoryStatusNotification packet byte	Description
8	Fault type; see Table 2-119 (page 155)
9	Fault condition; see Table 2-120 (page 155)

Table 2-119 Accessory fault types

Value	Description
0x00	No fault
0x01	Voltage fault
0x02	Current fault
0x03–0xFF	Reserved

Table 2-120 Accessory fault conditions

Value	Description
0x00	No fault
0x01	Recoverable
0x02	Not recoverable
0x03–0xFF	Reserved

[Table 2-121](#) (page 155) lists a sample command flow that uses the General lingo 0x46–0x48 accessory status notification commands.

Table 2-121 Sample IDPS process and accessory status notification commands

Step	Accessory command	Apple device command	Comment
1	StartIDPS		no params

Step	Accessory command	Apple device command	Comment
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> ■ If the accessory receives no ACK command within 500 ms, it may retry Step 1 at 500 ms intervals as long as Apple Device Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them. ■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the Apple device does not support IDPS. The accessory must not proceed to Step 3. Instead, it must perform either one of these two alternative actions: <ul style="list-style-type: none"> □ If IDPS support is required, within 800 ms the accessory must send an IdentifyDeviceLingoes command with all fields set to 0xFF. This causes the Apple device to display an "Accessory not supported" message. □ If IDPS support is not required, the accessory must then send another IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x00. See "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). A sample command sequence is listed in Table F-41 (page 621). ■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1. 			
3	SetFIDTokenValues		<p>setting FID tokens:</p> <p>IdentifyToken: General Lingo 00 device options: 02 (auth immediately); device id: 0200.</p> <p>AccCapsToken: line out.</p> <p>AccInfoToken: info type 04, Accessory Firmware Version: 1.0.0; info type 05, Accessory Hardware Version: 1.0.0; info type 06, Accessory Manufacturer: Apple; info type 07, Accessory Model Number: 1234; info type 08, Accessory Serial Number: 5678; info type 01, Accessory Name = Brabazon: 42 72 61 62 61 7A 6F 6E 00; info type 0B, Accessory Status Types supported: 00 00 00 02.</p>
4		RetFIDTokenValueACKS	returning acknowledgments for FID tokens
5	EndIDPS		terminating IDPS process
6		IDPSStatus	status 'ready for auth'

Step	Accessory command	Apple device command	Comment
7		GetDevAuthentication-Info	no params
The accessory proceeds with authentication, as illustrated in Table 2-5 (page 74).			
n		SetAccStatus-Notification	send mask value of 0x00000004 to set notifications of accessory fault conditions
n+1	RetAccStatus-Notification		send mask value of 0x00000004 to acknowledge that notifications of accessory fault conditions are set
n+2	AccessoryStatus-Notification		send data values of 0x00,0x00 to initialize fault monitoring in the Apple device to 'no fault' status and mode (see Table 2-119 (page 155) and Table 2-120 (page 155)).

Command 0x49: SetEventNotification

Direction: Accessory to Apple device

This command enables asynchronous remote event notifications for specific Apple device events, as described in [Apple Device Event Notifications](#) (page 523).

When the accessory identifies itself, it can check the attached Apple device's notification support by sending a `GetSupportedEventNotification` command. If the Apple device supports notifications, it returns a `RetSupportedEventNotification` command confirming that it supports at least Flow Control notifications (bit 2 in the 64-bit big-endian notification bitmask). If the accessory needs to receive notifications of types that the attached Apple device supports, it must then send this command to register for one or more of the notifications listed in [Table 2-123](#) (page 158).

The Apple device acknowledges this command with a General Lingo `ACK` command reporting Status OK (0x00).

If the accessory registers for camera notifications, the Apple device also sends an `iPodNotification` command reporting its current Camera mode. The Apple device may send these commands in any order.

For a suggested sequence of commands to enable Apple device notifications, see [Apple Device Event Notifications](#) (page 523).

Table 2-122 SetEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x0C	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x49	Command ID: SetEventNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command. If the Apple device does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0x00	Event notification mask (bits 63:56)
8	0x00	Event notification mask (bits 55:48)
9	0x00	Event notification mask (bits 47:40)
10	0x00	Event notification mask (bits 39:32)
11	0x00	Event notification mask (bits 31:24)
12	0x00	Event notification mask (bits 23:16)
13	0x00	Event notification mask (bits 15:8)
14	0xNN	Event notification mask (bits 7:0); at least bit 2 must be set (see Table 2-123 (page 158))
15	0xNN	Checksum

Table 2-123 Notification bitmask bits

Bit number	Description	Notes
0-1	Reserved; set to 0	
2	Flow control	Must always set to 1. If and only if the accessory uses IDPS, Flow Control notifications are enabled by default. See "Accessory Identification" (page 519).
3	Radio tagging status	See "iTunes Tagging" (page 535).
4	Camera status	See "Accessory Control of the iPod 5G nano Camera" (page 190).
5	Charging info	See "Command 0x54: SetAvailableCurrent" (page 180).
6-9	Reserved; set to 0	
10	NowPlayingFocusApp status	See "Accessory Launching of iOS Applications" (page 49).
11	SessionSpaceAvailable status	See Table 2-130 (page 163).

Bit number	Description	Notes
12	Database available	See Table 2-131 (page 163).
13	Command Completed	See Table 2-132 (page 164). This notification can be used to determine completion of the <code>CreateGeniusPlaylist</code> (Lingoes 0x03 and 0x04) and <code>RefreshGeniusPlaylist</code> (Lingo 0x04) commands, because they may take up to 30 seconds to finish.
14-63	Reserved; set to 0	

Command 0x4A: iPodNotification

Direction: Apple device to Accessory

If an accessory has registered for notifications, using `SetEventNotification`, the Apple device sends this command to the accessory every time there is a change in the state of any of the processes for which notification was requested. The notification timings and payloads are different for each of the notification types listed in [Table 2-123](#) (page 158):

- If Flow Control notifications are enabled, the Apple device sends a Flow Control notification whenever the incoming queue is full and the Apple device is unable to accept more packets. When an accessory receives this notification, it must stop sending packets to the Apple device for the wait time specified by the notification packet shown in [Table 2-124](#) (page 160). If transaction IDs are being used, the Flow Control notification also returns the transaction ID of the packet that caused the overflow. This packet was not added to the incoming data queue; hence the accessory can use the overflow transaction ID to determine which packets need to be resent after the wait time.
- If the accessory registers for Radio Tagging notifications, the Apple device sends notifications when tagging information is available or when a tagging operation is initiated (either from the Apple device's user interface or from the accessory via the Simple Remote lingo `RadioButtonStatus` command). The notification packet for Radio Tagging is shown in [Table 2-125](#) (page 161).
- If the accessory registers for Camera notifications, the Apple device sends a notification immediately after receiving `SetEventNotification` and subsequently whenever its Camera mode changes. It passes the state of its Camera mode using the packet format shown in [Table 2-127](#) (page 162). Because the Apple device's notification mechanism is asynchronous, the accessory may receive this packet before its `SetEventNotification` command has been acknowledged. The Apple device may also send notifications to the accessory at any time as the result of changes in the Camera application's state that may be unrelated to any action taken by the accessory. For full details see "[Accessory Control of the iPod 5G nano Camera](#)" (page 190).
- If the accessory registers for `NowPlayingFocusApp` notifications, the Apple device sends a notification when the currently active media playback app changes. The media playback app may be different than the app that manages the user interface. The notification packet for `NowPlayingFocusApp` is shown in [Table 2-129](#) (page 162).
- If the accessory registers for `SessionSpaceAvailable` notifications, the Apple device sends a notification when its data space for `DevDataTransfer` commands transitions from Full to Space Available. If the accessory receives an ACK status of `SessionWriteFailure` (0x17), it must wait for this notification

before retrying a `DevDataTransfer` command in the same session. The accessory should try sending a `DevDataTransfer` command with a small data payload at first, to avoid immediately generating another `SessionWriteFailure` error.

The accessory can check for the availability of this notification by sending a `GetSupportedEventNotification` command and checking whether bit position 11 is set in the returned `Event Notification Mask`. The accessory can determine whether this notification is set by sending a `GetEventNotification` command and checking the same bit. The notification packet for `SessionSpaceAvailable` is shown in [Table 2-130](#) (page 163).

- If the accessory registers for Command Complete notifications, the Apple device sends an asynchronous notification when it completes the command it is currently executing. An example is when the accessory has sent the Apple device an Extended Interface lingo `CreateGeniusPlaylist` command; the Apple device responds first with an ACK command to indicate that it is working on the playlist and then an `iPodNotification Command Complete` when it has finished. The notification packet for Command Complete is shown in [Table 2-132](#) (page 164).
- If the accessory registers for USB Host Mode Charging notifications, the Apple device sends a notification immediately after receiving `SetEventNotification` and subsequently whenever its charging info changes. See "[Command 0x54: SetAvailableCurrent](#)" (page 180). The Apple product passes the charging information using the packet format shown in [Table 2-133](#) (page 164).

Note: The Apple device may send `iPodNotification` commands at any time, including during the IDPS process. The accessory must not acknowledge these commands and must simply ignore any that it cannot handle. The Apple device does not retry these commands.

Table 2-124 `iPodNotification` packet for Flow Control Notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0B or 0x07	Length of packet payload: 0x0B if the Apple device supports IDPS, 0x07 otherwise.
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: <code>iPodNotification</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command. If the Apple device does not support IDPS, this and the next byte are omitted and the value of byte 2 is 0x07.
6	0xNN	<code>transID</code> [bits 7:0]
7	0x02	Notification type: Flow Control
8	0xNN	Wait time in ms (bits 31:24)
9	0xNN	Wait time in ms (bits 23:16)
10	0xNN	Wait time in ms (bits 15:8)

Byte number	Value	Comment
11	0xNN	Wait time in ms (bits 7:0)
12	0xNN	Overflow transaction ID (bits 15:8); Transaction ID of the packet that caused the Apple device's incoming data queue to overflow. If the Apple device does not support IDPS, this and the next byte are omitted.
13	0xNN	Overflow transaction ID (bits 7:0)
14	0xNN	Checksum

Table 2-125 iPodNotification packet for Radio Tagging Notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: iPodNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command. If the Apple device does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0x03	Notification type: Radio Tagging
8	0xNN	Tag status; see Table 2-126 (page 161).
9	0xNN	Checksum

Table 2-126 iPodNotification payload for Radio Tagging notifications

Byte value	Description
0x00	Tagging operation successful
0x01	Tagging operation failed
0x02	Information available for tagging (all required RT+ information has been received)
0x03	Information not available for tagging (tagging is not possible because the RT+ information has been reset due to a song change, station change, loss of signal, etc.)
0x04-0xFF	Reserved

Table 2-127 `iPodNotification` packet for Camera notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: <code>iPodNotification</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command. If the Apple device does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	<code>transID</code> [bits 7:0]
7	0x04	Notification type: Camera Notifications
8	0xNN	Payload; see Table 2-128 (page 162).
9	0xNN	Checksum

Table 2-128 `iPodNotification` payload for Camera Notifications

Byte value	Description
0x00	Camera App Off
0x01-0x02	Reserved
0x03	Preview
0x04	Recording
0x05-0xFF	Reserved

Table 2-129 `iPodNotification` packet for `NowPlayingFocusApp` notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: <code>iPodNotification</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command

Byte number	Value	Comment
6	0xNN	transID [bits 7:0]
7	0x0A	Notification type: NowPlayingAppFocusChange
8-NN	0xNN	Null-terminated UTF-8 string containing the Application ID of the application that has just now received playing focus.
Last byte	0xNN	Checksum

Table 2-130 iPodNotification packet for Session Space Available notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: iPodNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command
6	0xNN	transID [bits 7:0]
7	0x0B	Notification type: SessionSpaceAvailable
8	0xNN	SessionID [bits 15:8]: ID of session with data space now available
9	0xNN	SessionID [bits 7:0]
10	0xNN	Checksum

Table 2-131 iPodNotification packet for Database Available notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: iPodNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command
6	0xNN	transID [bits 7:0]

Byte number	Value	Comment
7	0x0C	Notification type: Database Available
8	0xNN	Database available status: 1 = available, 0 = not available. Accessories must not access the database or enter Extended Interface or iPod Out modes while the database is unavailable. If the Apple device was previously in Extended Interface or iPod Out mode, the accessory must send <code>EnterRemoteUIMode</code> or <code>SetUIMode</code> to explicitly re-enter the previous mode when notified that the database has become available.
9	0xNN	Checksum

Table 2-132 `iPodNotification` packet for Command Complete notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x09	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: <code>iPodNotification</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command
6	0xNN	<code>transID</code> [bits 7:0]
7	0x0D	Notification type: Command Complete
8	0xNN	LingoID: Lingo ID of the command that has completed
9	0xNN	<code>CmdID</code> [bits 15:8]: ID of the command that has completed
10	0xNN	<code>CmdID</code> [bits 7:0]
11	0xNN	Command completion status: 0x00: Command successful 0x01: Command failed 0x02: Command cancelled 0x03-0xFF: Reserved
12	0xNN	Checksum

Table 2-133 `iPodNotification` packet for USB Host mode charging notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0x08	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: <code>iPodNotification</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command
6	0xNN	<code>transID</code> [bits 7:0]
7	0x05	Notification type: Charging Info
8	0xNN	Charging Info type: see Table 2-134 (page 165).
9	0xNN	Charging Info value (bits 15:8): see Table 2-135 (page 165)
10	0xNN	Charging Info value (bits 7:0)
11	0xNN	Checksum

Table 2-134 InfoType values for USB Host mode charging notifications

Value	Meaning
0x00	Available current
0x01-0xFF	Reserved

Table 2-135 Charging Info values for `iPodNotification`

Value	Bytes	Meaning
0x00	2	Available current: specifies the Apple device's new absolute current-sink limit setting in mA.
0x01-0xFF	Reserved	

Command 0x4B: `GetiPodOptionsForLingo`

Direction: Accessory to Apple device

The accessory sends this command to ask the Apple device to return a 64-bit field that defines the options that the Apple device supports for a specific lingo, identified in `LingoID`. In response, the Apple device sends a `RetiPodOptionsForLingo` command.

If the accessory requests options for the General lingo (0x00) and the Apple device returns an ACK with nonzero status, then the Apple device does not support `GetiPodOptionsForLingo`; the accessory should use `RequestLingoProtocolVersion` instead. If the accessory requests options for any other lingo and the Apple device returns a nonzero ACK, that lingo is not listed in [Table 2-138](#) (page 167) or is not supported by the Apple device on the port being used; no `RetiPodOptionsForLingo` command will be returned.

Table 2-136 `GetiPodOptionsForLingo` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x00	Lingo ID: General lingo
4	0x4B	Command ID: <code>GetiPodOptionsForLingo</code>
5	0xNN	LingoID: ID of lingo for which options are requested; see Table 2-138 (page 167).
6	0xNN	Checksum

Command 0x4C: `RetiPodOptionsForLingo`

Direction: Apple device to Accessory

The Apple device sends this command in response to a `GetiPodOptionsForLingo` command from the accessory. It returns the ID of the lingo for which options were requested in `LingoID` and the option bits as a 64-bit big-endian field in bytes 6-13. [Table 2-138](#) (page 167) lists the available option bit values for various lingoes, and [Table 2-139](#) (page 170) lists a sample command flow that queries an Apple device's options for those lingoes.

Table 2-137 `RetiPodOptionsForLingo` packet

Byte number	Value	Comment	
0	0xFF	Sync byte (required only for UART serial)	
1	0x55	Start of packet (SOP)	
2	0x0B	Length of packet	
3	0x00	Lingo ID: General lingo	
4	0x4C	Command ID: <code>RetiPodOptionsForLingo</code>	
5	0xNN	LingoID: ID of lingo for which options were requested.	
6	0xNN	Option bits (bits 63:56)	See Table 2-138 (page 167)
7	0xNN	Option bits (bits 55:48)	

Byte number	Value	Comment
8	0xNN	Option bits (bits 47:40)
9	0xNN	Option bits (bits 39:32)
10	0xNN	Option bits (bits 31:24)
11	0xNN	Option bits (bits 23:16)
12	0xNN	Option bits (bits 15:8)
13	0xNN	Option bits (bits 7:0)
14	0xNN	Checksum

Table 2-138 RetiPodOptionsForLingo option bits

Lingo	Lingo ID	Bit	Description
General	0x00	00	Line out usage
		01	Video output
		02	NTSC video signal format
		03	PAL video signal format
		04	Composite video out connection
		05	S-Video video out connection
		06	Component video out connection
		07	Closed Captioning (video)
		08	Video aspect ratio 4:3 (fullscreen)
		09	Video aspect ratio 6:9 (widescreen)
		10	Subtitles (video)
		11	Video Alternate Audio Channel
		12	Reserved
		13	Communication with iOS applications
		14	Apple device notifications
		15-18	Reserved
		19	Pause on power removal preference control

Lingo	Lingo ID	Bit	Description
		20-22	Reserved
		23	0 = restricted IDPS token handling. If this bit is 0, the Apple device's IDPS process can handle only <code>IdentifyToken</code> , <code>AccCapsToken</code> , <code>AccInfoToken</code> , <code>iPodPreferenceToken</code> , <code>EAPProtocolToken</code> , and <code>BundleSeedIDPrefToken</code> . See Table 2-76 (page 131).
		24	Request Application Launch; see "Accessory Launching of iOS Applications" (page 49).
		25	Reserved
		26	1 = The Apple device supports switching between Apple device analog line out and USB digital audio output routings using the General lingo <code>SetiPodPreference</code> line out preference. This must be used by USB host accessories with Apple devices that don't support the Digital Audio lingo <code>SetVideoDelay</code> command for maintaining audio/video stream synchronization when video media is playing See "Audio/Video Synchronization" (page 351). 0 = The Apple device does not support switching audio routings using the <code>SetiPodPreference</code> command. To switch between USB digital audio and analog line out audio routings, the accessory must identify and authenticate itself repeatedly with and without the Digital Audio lingo.
		27	USB Host Mode capable (using 28 k Ω accessory ID resistor)
		28	USB Host Mode supports audio output streaming to USB audio devices; valid only if bit 27 is set.
		29	USB Host Mode supports audio input streaming from USB audio devices; valid only if bit 27 is set.
		30	USB Host Mode does not support maximum current draw in Low Power Mode; see "Low Power Mode" in <i>MFi Accessory Hardware Specification</i> , release R7 or later.
		31-63	Reserved
Simple Remote	0x02	00	Context-specific controls
		01	Audio media controls
		02	Video media controls
		03	Image media controls
		04	Sports media controls
		05-07	Reserved
		08	Camera media controls

Lingo	Lingo ID	Bit	Description
		09	USB HID commands
		10	Accessibility controls
		11	Accessibility preferences
		12-63	Reserved
Display Remote	0x03	00	UI Volume control
		01	Absolute Volume control
		02	Supports Genius Playlist creation
		03-63	Reserved
Extended Interface	0x04	00	Video browsing
		01	The following Extended Interface commands are enabled: GetDBiTunesInfo RetDBiTunesInfo GetUIDTrackInfo RetUIDTrackInfo GetDBTrackInfo RetDBTrackInfo GetPBTrackInfo RetPBTrackInfo
		02	Nested playlists
		03	Supports Genius Playlist creation and refresh
		04	Apple device displays images sent by Extended Interface command " Command 0x0032: SetDisplayImage " (page 485).
		05	The accessory can access the list of categories that the he Apple device supports
		06	Supports <code>PlayControl</code> Play and Pause commands
		07-63	Reserved
Accessory Power	0x05	00-63	Reserved
USB Host Mode	0x06	00	Mode invoked by hardware; see <i>MFi Accessory Hardware Specification</i> , Release R4 or later.
		01	Mode invoked by firmware; see " Using an Apple Device as a USB Host " (page 54).

Lingo	Lingo ID	Bit	Description
		02-63	Reserved
RF Tuner	0x07	00	RDS Raw Mode support
		01	HD Radio Tuning support
		02	AM Radio Tuning support
		03-63	Reserved
Accessory Equalizer	0x08	00-63	Reserved
Sports	0x09	00	Reserved
		01	Nike + iPod Cardio Equipment
		02-63	Reserved
Digital Audio	0x0A	00	A/V Synchronization: 1=SetVideoDelay enabled, 0=SetVideoDelay disabled. See "Audio/Video Synchronization" (page 351), Step 3.
		01-63	Reserved
Storage	0x0C	00	iTunes Tagging
		01	Nike + iPod Cardio Equipment
		02-63	Reserved
iPod Out	0x0D	00	iPod Out is available
		01-63	Reserved
Location	0x0E	00	The Apple device accepts NMEA GPS location data
		01	The Apple device can send location assistance data
		02-63	Reserved

Table 2-139 (page 170) lists a sample command flow that queries an Apple device's options for most of the iAP lingoes.

Table 2-139 Sample GetiPodOptionsForLingo and RetiPodOptionsForLingo commands

Step	Accessory command	Apple device command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'

Step	Accessory command	Apple device command	Comment
<p>If the ACK reply to <code>StartIDPS</code> returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> ■ If the accessory receives no ACK command within 500 ms, it may retry Step 1 at 500 ms intervals as long as Apple Device Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them. ■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the iPod does not support IDPS. The accessory must not proceed to Step 3. Instead, it must perform either one of these two alternative actions: <ul style="list-style-type: none"> □ If IDPS support is required, within 800 ms the accessory must send an <code>IdentifyDeviceLingoes</code> command with all fields set to 0xFF. This causes the iPod to display an "Accessory not supported" message. □ If IDPS support is not required, the accessory must then send another <code>IdentifyDeviceLingoes</code> command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x00. See "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). A sample command sequence is listed in Table F-41 (page 621). ■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1. 			
3	<code>GetiPodOptions-ForLingo</code>		getting options for General Lingo
4		<code>RetiPodOptions-ForLingo</code>	returning options of 0000000000006FFF (Line out usage Video output NTSC video signal format PAL video signal format Composite video out connection S-Video video out connection Component video out connection Closed captioning (video) Video aspect ratio 4:3 (fullscreen) Video aspect ratio 6:9 (widescreen) Subtitles (video) Video Alternate Audio Channel App communication capable iPod notifications) for General Lingo
5	<code>GetiPodOptions-ForLingo</code>		getting options for Simple Remote Lingo
6		<code>RetiPodOptions-ForLingo</code>	returning options of 000000000000000F (Audio media controls Video media controls Image media controls context-specific controls) for Simple Remote Lingo

Step	Accessory command	Apple device command	Comment
7	GetiPodOptions-ForLingo		getting options for Display Remote Lingo
8		RetiPodOptions-ForLingo	returning options of 0000000000000001 (Volume control) for Display Remote Lingo
9	GetiPodOptions-ForLingo		getting options for Extended Interface Lingo
10		RetiPodOptions-ForLingo	returning options of 0000000000000001 (Video browsing) for Extended Interface Lingo
11	GetiPodOptions-ForLingo		getting options for Accessory Power Lingo
12		RetiPodOptions-ForLingo	returning options of 0000000000000000 (none) for Accessory Power Lingo
13	GetiPodOptions-ForLingo		getting options for Accessory Equalizer Lingo
14		RetiPodOptions-ForLingo	returning options of 0000000000000000 (none) for Accessory Equalizer Lingo
15	GetiPodOptions-ForLingo		getting options for Sports Lingo
16		RetiPodOptions-ForLingo	returning options of 0000000000000002 (Nike + iPod cardio equipment) for Sports Lingo
17	GetiPodOptions-ForLingo		getting options for Digital Audio Lingo
18		RetiPodOptions-ForLingo	returning options of 0000000000000000 (none) for Digital Audio Lingo
19	GetiPodOptions-ForLingo		getting options for Storage Lingo
20		RetiPodOptions-ForLingo	returning options of 0000000000000003 (iTunes tagging Nike + iPod cardio equipment) for Storage Lingo
21	GetiPodOptions-ForLingo		getting options for Location Lingo

Step	Accessory command	Apple device command	Comment
22		RetiPodOptions-ForLingo	returning options of 0000000000000003 (Apple device accepts NMEA GPS location data Apple device can send location assistance data) for Location Lingo
23	SetFIDTokenValues		setting 8 FID tokens ; AccInfoToken = Acc name (Apple device Accessory Name); AccInfoToken = Acc FW version (v1.2.1); AccInfoToken = Acc HW version (v1.0.0); AccInfoToken = Acc manufacturer (Apple Inc.); AccInfoToken = Acc model number (MA1390LL/A); EAProtocolToken = 1 (com.apple.protocolMain); EAProtocolToken = 2 (com.apple.protocolAlternative); BundleSeedIDString = 24D4XFAF43
24		RetFIDTokenValueACKs	8 ACKs for FID tokens ; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; EAProtocolToken = (1) accepted; EAProtocolToken = (2) accepted; BundleSeedIDPrefToken = accepted
25	SetFIDTokenValues		setting 1 FID tokens ; IdentifyToken = (lingoes: 0/2 options 0x00000006 device ID 0x00000200)
26		RetFIDTokenValueACKs	1 ACKs for FID tokens ; IdentifyToken = accepted
27	SetFIDTokenValues		setting 1 FID tokens ; AccCapsToken = 0x00000000000000A17
28		RetFIDTokenValueACKs	1 ACKs for FID tokens ; AccCapsToken = accepted
29	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'selected')
30		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (line out usage) accepted

Step	Accessory command	Apple device command	Comment
31	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'on' for preference class 'video out setting' with restore on exit 'selected')
32		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (video out setting) accepted
33	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'widescreen' for preference class 'screen configuration' with restore on exit 'selected')
34		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (screen configuration) accepted
35	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'PAL' for preference class 'video format setting' with restore on exit 'selected')
36		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (video format setting) accepted
37	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'component' for preference class 'video connection' with restore on exit 'selected')
38		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (video connection) accepted
39	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting '16:9' for preference class 'aspect ratio' with restore on exit 'selected')
40		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (aspect ratio) accepted
41	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'component' for preference class 'video connection' with restore on exit 'selected')

Step	Accessory command	Apple device command	Comment
42		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (video connection) accepted
43	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'on' for preference class 'video subtitles' with restore on exit 'selected')
44		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (video subtitles) accepted
45	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'on' for preference class 'alternate audio channel' with restore on exit 'selected')
46		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (alternate audio channel) accepted
47	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'on' for preference class 'closed captions' with restore on exit 'selected')
48		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (closed captions) accepted
49	EndIDPS		status 'finished with IDPS; proceed to authentication'
50		IDPSStatus	status 'ready for auth'
51		GetDevAuthentication-Info	no params
52	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1; cert data: ...
53		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
54	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1; cert data: ...
55		AckDevAuthentication-Info	acknowledging 'auth info supported'

Step	Accessory command	Apple device command	Comment
56		GetDevAuthentication-Signature	offering challenge '...' with retry counter 1
57	RetDevAuthentication-Signature		returning signature '...'
58		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'

Command 0x4D: GetEventNotification

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to ask for the Apple device's current 64-bit big-endian notification bitmask.

Table 2-140 GetEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4D	Command ID: GetEventNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command. If the Apple device does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0xNN	Checksum

Command 0x4E: RetEventNotification

Direction: Apple device to Accessory

The Apple device sends this command to the accessory in response to a GetEventNotification command. It passes its current 64-bit big-endian notification bitmask.

Table 2-141 RetEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0C	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4E	Command ID: RetEventNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command. If the Apple device does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0x00	Event notification mask (bits 63:56)
8	0x00	Event notification mask (bits 55:48)
9	0x00	Event notification mask (bits 47:40)
10	0x00	Event notification mask (bits 39:32)
11	0x00	Event notification mask (bits 31:24)
12	0x00	Event notification mask (bits 23:16)
13	0x00	Event notification mask (bits 15:8)
14	0xNN	Event notification mask (bits 7:0); see Table 2-123 (page 158).
15	0xNN	Checksum

Command 0x4F: GetSupportedEventNotification

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to query the Apple device's support for notifications. The Apple device responds with a RetSupportedEventNotification command.

Table 2-142 GetSupportedEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload

Byte number	Value	Comment
3	0x00	Lingo ID: General lingo
4	0x4F	Command ID: GetSupportedEventNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command. If the Apple device does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0xNN	Checksum

Command 0x50: CancelCommand

Direction: Accessory to Apple device

This command lets an accessory cancel a multipacket data transfer request that it has sent to an Apple device. It requires that the accessory use unique transaction IDs for its commands and that it has declared its ability to handle multipacket downloads by setting bit 19 in the IDPS Accessory capabilities token; see [Table 2-80](#) (page 133). This command may be used to cancel only the following commands:

Simple Remote lingo GetCurrentItemProperty command

Display Remote lingo GetTrackArtworkData command

Extended Interface lingo GetTrackArtworkData command

Extended Interface lingo RetrieveCategorizedDatabaseRecords command

If the CancelCommand is successful, the Apple device responds with an ACK command passing 0x00 and the CancelCommand's transaction ID. A typical use for CancelCommand is to halt a long download of artwork from the Apple device if the user switches to a different track.

Table 2-143 CancelCommand packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x50	Command ID: CancelCommand
5	0xNN	transID [bits 15:8]: Transaction ID of this CancelCommand command; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	LingoID: ID of lingo of command being cancelled

Byte number	Value	Comment
8	0xNN	CommandID [bits 15:8]: Command ID of command being cancelled.
9	0xNN	CommandID [bits 7:0]
10	0xNN	TransactionID [bits 15:8]: Transaction ID of command being cancelled.
11	0xNN	TransactionID [bits 7:0]
12	0xNN	Checksum

Command 0x51: RetSupportedEventNotification

Direction: Apple device to Accessory

The Apple device sends this command to the accessory in response to a `GetSupportedEventNotification` command. It passes a 64-bit big-endian bitmask that defines the types of notifications that the Apple device supports.

Table 2-144 RetSupportedEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0C	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x51	Command ID: RetSupportedEventNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command. If the Apple device does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0x00	Event notification mask (bits 63:56)
8	0x00	Event notification mask (bits 55:48)
9	0x00	Event notification mask (bits 47:40)
10	0x00	Event notification mask (bits 39:32)
11	0x00	Event notification mask (bits 31:24)
12	0x00	Event notification mask (bits 23:16)
13	0x00	Event notification mask (bits 15:8)

Byte number	Value	Comment
14	0xNN	Event notification mask (bits 7:0); see Table 2-123 (page 158).
15	0xNN	Checksum

Command 0x54: SetAvailableCurrent

Direction: Device to iPod

If the accessory is designed to provide power to an Apple device, it sends this command to set the absolute current-sink limit that the Apple device may draw. Without this command, the default value is 500 mA. The accessory may either raise or lower the limit.

If 5 V is present on USB V_{BUS} (pin 8 of the 30-pin connector), and if the limit set by this command is higher than the existing limit, the Apple device may immediately switch to the new limit.

Whether the limit is raised or lowered, the Apple device will send an `iPodNotification` command after it has updated its current-sink limit value. To receive this notification, the accessory must have previously registered for Charging Info notifications by sending a General lingo `SetEventNotification` command that sets bit 5, as listed in [Table 2-123](#) (page 158).

Table 2-145 SetAvailableCurrent packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial; omit byte when sending by USB)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x00	Lingo ID: General lingo
4	0x54	Command ID: SetAvailableCurrent
5	0xNN	CurrentLimit [bits 15:8]: Maximum current in mA that the Apple device may draw.
6	0xNN	CurrentLimit [bits 7:0]
7	0xNN	Checksum

The following sequence is a typical example of using this command:

1. The Apple device is plugged into a dock that is powered by a power brick.
2. The dock reads the brick's D+/D− resistors, which specify its maximum available charging current.
3. The dock reduces the available current value by the amount of current it requires for its own use.

4. The dock sends a `SetAvailableCurrent` command to the Apple device, passing the reduced value, to specify the maximum current that the Apple device may draw.

Command 0x64: RequestApplicationLaunch

Direction: Accessory to Apple device

The accessory sends this command to an Apple device to request that it launch a specific application. The accessory passes an Application Bundle ID string, such as `"com.mycompany.myapp"`, to specify which application to launch. The Apple device replies with an ACK command that returns the application launch status.

Note: An ACK status of OK (0x00) only indicates that the launch is possible; it does not guarantee that the application is running at that time. A status of Command Failed indicates that the application either does not exist on the Apple device or that the Apple device is in a condition that prevents the launch. In this case, the Accessory must not retry the `RequestApplicationLaunch` command.

Table 2-146 RequestApplicationLaunch packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x64	Command ID: RequestApplicationLaunch
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0x00	Reserved
8	0x01	Reserved; must be set to 1.
9	0x00	Reserved
10-0xNN	0xNN	Null-terminated UTF-8 string containing the Application ID of the application to be launched.
Last byte	0xNN	Checksum

Command 0x65: GetNowPlayingFocusApp

Direction: Accessory to Apple device

An accessory sends this command to an Apple device to determine which application has the current Now Playing focus (the actual playing of media, which may be different from the user interface focus). The Apple device responds with a `RetNowPlayingFocusApp` command.

Table 2-147 `GetNowPlayingFocusApp` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x65	Command ID: <code>GetNowPlayingFocusApp</code>
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	Checksum

Command 0x66: `RetNowPlayingFocusApp`

Direction: Apple device to Accessory

An Apple device sends this command to an accessory, in response to a `GetNowPlayingFocusApp` command, to report which application has the current Now Playing focus.

Table 2-148 `RetNowPlayingFocusApp` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x66	Command ID: <code>RetNowPlayingFocusApp</code>
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7-0xNN	0xNN	Null-terminated UTF-8 string containing the Application ID of the application that has the current Now Playing focus.
Last byte	0xNN	Checksum

Accessory Lingo

The iPod Accessory Protocol (iAP) defines a number of different accessory lingoes. This chapter describes these lingoes and their commands.

[Table 3-1](#) (page 183) shows the available accessory lingoes and their IDs.

Table 3-1 iPod accessory lingoes

Lingo	ID	Notes
Microphone	0x01	Deprecated; see " Deprecated Lingo 0x01: Microphone Lingo " (page 600).
Simple Remote	0x02	
Display Remote	0x03	
Extended Interface	0x04	See The Extended Interface Protocol (page 409). For a sample command sequence that declares the Extended Interface lingo, see Table 3-310 (page 403).
Accessory Power	0x05	
USB Host Mode	0x06	
RF Tuner	0x07	
Accessory Equalizer	0x08	
Sports	0x09	
Digital Audio	0x0A	
Reserved	0x0B	
Storage	0x0C	
iPod Out	0x0D	
Location	0x0E	
Reserved	0x0F–0xFF	

Command Timings

Some iAP commands sent by Apple devices take a significant time to execute, and some of these retry if the accessory does not acknowledge the first attempt. Timeout information and the number of times the command tries to execute are shown in [Table 3-2](#) (page 184).

Table 3-2 Select Apple device command timings

Lingo	Command	Timeout	Tries
RF Tuner (0x07) (See note below)	GetCaps (0x01)	200 ms	1
	GetTunerCtrl (0x03)	200 ms	1
	SetTunerCtrl (0x05)	200 ms	1
	GetTunerBand (0x06)	200 ms	1
	SetTunerBand (0x08)	200 ms	1
	GetTunerFreq (0x09)	200 ms	1
	SetTunerFreq (0x0B) (AM mode)	300 ms	1
	SetTunerFreq (0x0B) (FM mode)	200 ms	1
	GetTunerMode (0x0C)	200 ms	1
	SetTunerMode (0x0E)	200 ms	1
	GetTunerSeekRssi (0x0F)	200 ms	1
	SetTunerSeekRssi (0x11)	200 ms	1
	TunerSeekStart (0x12)	200 ms	1
	GetTunerStatus (0x14)	200 ms	1
	GetStatusNotifyMask (0x16)	200 ms	1
	SetStatusNotifyMask (0x18)	200 ms	1
	GetRdsReadyStatus (0x1A)	200 ms	1
	GetRdsData (0x1C)	200 ms	1
	GetRdsNotifyMask (0x1E)	200 ms	1
	SetRdsNotifyMask (0x20)	200 ms	1
Accessory Equalizer (0x08)	GetCurrentEQIndex (0x01)	2500 ms	3
	SetCurrentEQIndex (0x03)	3000 ms	3

Lingo	Command	Timeout	Tries
	GetEQSettingCount (0x04)	3000 ms	3
	GetEQIndexName (0x06)	3000 ms	3
Sports (0x09)	GetDeviceCaps (0x03)	500 ms	2
Digital Audio (0x0A)	GetAccSampleRateCaps (0x02)	3000 ms	3
Location (0x0E)	GetDevCaps (0x01)	500 ms	1
	GetDevControl (0x03)	500 ms	1
	SetDevControl (0x05)	500 ms	1
	GetDevData (0x06)	500 ms	1
	SetDevData (0x08)	500 ms	1

RF Tuner lingo note: The accessory must not exceed the RF Tuner command timings listed when responding to commands from an Apple device.

Lingo 0x02: Simple Remote Lingo

This lingo is intended for a remote accessory that retains no state information about the Apple device. Simple commands are sent to the Apple device, and no acknowledgment or state information is sent back to the accessory. This lingo is used by the Apple device's standard in-line remote control. For a sample command sequence that declares this lingo, see [Table 3-311](#) (page 404).

History and Applicability

System software versions 2.0 through 2.2 on the 3G iPod and versions 1.0 and 1.1 of the iPod mini support only the first five button responses (0 through 4) in the `ContextButtonStatus` command. This is called the contextual button lingo and includes only command 0x00. An extended set of commands (0x01 through 0x04) is available in the dedicated media lingoes, as shown in [Table 3-3](#) (page 185).

Table 3-3 Simple remote lingo command summary

Command	ID	Direction	Packet	Lingo version	Authentication
0x00	ContextButtonStatus	Acc to Dev	buttonStatus:4	All	No
0x01	ACK	Dev to Acc	cmdStatus:1, cmdID:1	1.01	Yes

Command	ID	Direction	Packet	Lingo version	Authentication
0x02	ImageButtonStatus	Deprecated; see " Deprecated Simple Remote Lingo Command " (page 611).			
0x03	VideoButtonStatus	Acc to Dev	buttonStatus:4	1.01	Yes
0x04	AudioButtonStatus	Acc to Dev	buttonStatus:4	1.01	Yes
0x05–0x0A	Reserved	N/A	N/A	N/A	N/A
0x0B	iPodOutButtonStatus	Acc to Dev	{buttonSource;1, iPodOutButtonMask;<var>}	1.03	Yes
0x0C	RotationInputStatus	Acc to Dev	{rotationSource;1, controllerType;1, rotationDirection;1, rotationAction;1} [{rotationType;1, detentsOrDegreesMoved;2, detentsOrDegreesTotal;2}]	1.03	Yes
0x0D	RadioButtonStatus	Acc to Dev	buttonStatus:1	1.02	Yes
0x0E	CameraButtonStatus	Acc to Dev	buttonStatus:1	1.02	Yes
0x0F	RegisterDescriptor	Acc to Dev	index:1, VID:2, PID:2, country_code:1, descriptor:<var>	1.04	Yes
0x10	SendHIDReportToiPod	Acc to Dev	index:1, report_type:1, report:<var>	1.04	Yes
0x11	SendHIDReportToAcc	Dev to Acc	index:1, report_type:1, report:<var>	1.04	Yes
0x12	UnregisterDescriptor	Acc to Dev	index:1	1.04	Yes
0x13	AccessibilityEvent	Acc to Dev	{eventType;1, eventData;<var>}	1.04	Yes
0x14	GetAccessibility-Parameter	Acc to Dev	paramType;1	1.04	Yes
0x15	RetAccessibility-Parameter	Dev to Acc	{paramType;1, paramValue;<var>}	1.04	Yes

Command	ID	Direction	Packet	Lingo version	Authentication
0x16	SetAccessibility-Parameter	Acc to Dev	{paramType;1, paramValue;<var>}	1.04	Yes
0x17	GetCurrentItem-Property	Acc to Dev	propertyType;1	1.04	Yes
0x18	RetCurrentItem-Property	Dev to Acc	{propertyType;1, propertyValue;<var>}	1.04	Yes
0x19	SetContext	Acc to Dev	context;1	1.04	Yes
0x1A–0x80	Reserved	N/A	N/A	N/A	N/A
0x81	DevACK	Acc to Dev	{cmdStatus:1, cmdIDorig:1}	1.04	Yes
0x82–0xFF	Reserved	N/A	N/A	N/A	N/A

Dedicated media lingo is supported by version 1.01 of the Simple Remote lingo. By using `GetPodOptionsForLingo` (or querying the version number, as shown in [Table 3-4](#) (page 187) if the Apple device does not support `GetPodOptionsForLingo`), an accessory can determine the level of command support.

Table 3-4 Simple remote lingo support versions

Version	Support
No version	Command 0 supported, buttons 0–4
No version	Command 0 supported, buttons 0–25
1.00	Version number can be obtained
1.01	Commands 0–4 supported, buttons 0–25, media controls
1.02	Bug fix for compatibility with some accessories: when an accessory identifies itself using the General lingo <code>Identify</code> command, the 200 ms time limit during which <code>ContextButtonStatus</code> must report that all buttons are up is removed.
1.03	iPod Out support.

[Table 3-5](#) (page 187) shows the history of command changes in the Simple Remote lingo:

Table 3-5 Simple Remote lingo command history

Lingo version	Command changes	Features
No version	Add: 0x00	Context specific button status, buttons 0–4

Lingo version	Command changes	Features
No version	None	Context-specific button status, buttons 5-25 added (4G iPod with firmware versions 3.0.0 and 3.0.1)
1.00	None	Version number available through <code>RequestLingoProtocol - Version</code>
1.01	Add: 0x01–0x04	Image-, video-, and audio-specific media control button status
1.02	Add: 0x0D, 0x0E	All-buttons-up timeout applied to all commands, except not to <code>ContextButtonStatus</code> when that command is used with the General lingo (0x00) <code>Identify</code> command (0x01).
1.04	Add: 0x0B, 0x0C, 0x0F–0x19, 0x81	Support USB Human Interface Accessory (HID) descriptors, support iPod Out, support user interface accessibility commands

Playback Engine Playlists

An Apple device's **playback engine** contains a list of queued and currently playing media tracks. Its contents can be queried via the Display Remote or Extended Interface lingo to obtain track information such as track name, artist, album, genre, etc. One way the playback engine can be loaded is for the user to traverse the Apple device UI, selecting one or more media menu items (such as music or videos) and pressing the play/pause button. Another way is for an accessory to send Extended Interface lingo commands to select database categories and initiate the playback of specific selections.

The list of media tracks currently queued for playback in the Apple device's playback engine is called the **Now Playing** list. The playlist consisting of all tracks for a given media type, such as audio or video, is called the **All Tracks** list.

Using Contextual Buttons

A simple remote accessory sends a `ContextButtonStatus` command to provide updated status on which buttons are held down or released. The data of the packet is a number of bytes indicating which buttons are currently held down. The bytes are constructed by ORing the masks of the buttons together. To indicate all buttons are released, the accessory must send a full data payload consisting of 0x00. While any buttons are held down, the accessory must periodically send an updated `ContextButtonStatus` packet on a 30 ms to 100 ms interval.

It is not necessary to transmit any trailing bytes in which no bits are set. If this option is exercised, the length of the packet in the header must be adjusted accordingly; that is, the packet payload length must be decreased to exclude the trailing zero byte(s) that will not be transmitted.

When the user presses and holds down a button, a simple remote accessory must generate the button status packet immediately and repeat it every 30 to 100 ms for as long as the button is pressed. If a second button is pressed while the first button is down, the button status packet sent by the accessory must include status for both buttons, and this packet must be repeated every 30 to 100 ms for as long as both buttons are held down. [Table 3-15](#) (page 199) lists the possible Apple device button states.

The Next Track and Previous Track commands skip to the next or previous track. If the current track has played less than two seconds, Previous Track backs up to the beginning of the previous track. If the current track has played more than two seconds, it backs up to the beginning of the current track. These commands skip to the next or previous track even when the current track has chapters.

If the current track is an audiobook or a podcast with chapters, the Next Chapter and Previous Chapter commands skip to the next or previous chapter; otherwise they have no effect. If a track has chapters and has played more than two seconds of the current chapter, the Previous Chapter command backs up to the beginning of the current chapter; if the track has played less than two seconds, it backs up to the beginning of the previous chapter.

Some Apple device button states are interpreted differently by the Apple device when pressed and held down for 2 seconds or more. These are as follows:

- The Next Track button is treated as a Scan Forward button when pressed and held while a track is playing.
- The Previous Track button is treated as a Scan Backward button when pressed and held while a track is playing.
- The Play/Pause button is treated as a Power Off button when pressed and held.
- In Apple devices before the 4G iPod (color display), the Menu button acted as a Display Backlight On/Off button when pressed and held. Starting with the 4G iPod (color display), pressing and holding the Menu button causes a jump to the top level menu.
- In certain older Apple devices, if the Apple device was in Browse mode the Select button was treated as an Add Track to On-The-Go Playlist button when pressed and held. This behavior is no longer supported.

Repeated Next Track and Previous Track commands (see [Table 3-15](#) (page 199)), without an intervening button status packet indicating all buttons are up, are interpreted as Fast Forward and Rewind commands. For a locking Fast Forward or Rewind button, use the Begin Fast Forward or Begin Rewind commands to start the operation and a Play/Resume command to return to the play state.

The Next and Previous Album commands (see [Table 3-15](#) (page 199)) have no effect if there is no next or previous album to go to in the Now Playing list. Similarly, the Next and Previous Chapter commands have no effect if the currently playing track does not have two or more chapters.

If the list of media tracks currently in the playback engine contains two or more different playlists (including the All Tracks playlist), the Next and Previous Playlist commands navigate these playlists. If the list of media tracks is from a single playlist, or none of the tracks are associated with a playlist, then the Next and Previous Playlist commands have no effect.

Use the following steps to wake an Apple device when a simple remote button is pressed (UART serial port only):

1. Send a 0xFF sync byte.
2. Wait 20 ms.
3. Send the button status packet.
4. Wait 30 to 100 ms.
5. Repeat steps 3 and 4 for as long as any button is pressed.

Multiple button status packets cannot be sent back to back; otherwise, the repeated button status packets may be misinterpreted as being part of a corrupted packet. Repeated packets must always be separated by a gap of more than 25 ms, so the Apple device knows that the new packet is not part of the previous packet (which may happen if the SYNC and SOP bytes in the first packet have been lost).

Using Dedicated Media Buttons

The iAP contextual button protocol sends command packets with button messages that are interpreted based on the Apple device user interface context. When an Apple device is playing media types such as images and video, however, remote controls can become overloaded and their behavior may become confusing to the Apple device user. In this case, the accessory should use dedicated media control button commands.

The dedicated media lingo includes an ACK command, so devices know that a packet has been received, and dedicated button status commands for each media type currently supported by Apple devices: images, slideshows, videos, and audio. Use of the dedicated media lingo requires accessory authentication.

Media control button status bits are organized so that the most frequently used buttons are assigned low bit positions. This reduces the button status packet sizes for frequently used buttons. Button status is maintained separately for all ports and all commands. This means that buttons can be in different states for different media control types.

Note: For a given port and media control type (except `ContextButtonStatus` from an accessory using the `Identify` command), if a command has not been received within approximately 200 ms after the last button status command, the button status will be reset to all buttons up.

Accessory Control of the iPod 5G nano Camera

This section describes how accessories can control the video camera in the iPod 5G nano, using the `CameraButtonStatus` command with General lingo notification commands.

5G nano Camera Modes

The camera in the iPod 5G nano is operated by a firmware application that can be on or off. In its Preview mode, the image that the camera captures is continuously displayed on the iPod's screen. Its Recording mode captures the video being previewed. The user can make each of three modes transition to other modes by means of user controls on either the iPod or its attached accessory, as shown in [Table 3-6](#) (page 190).

Table 3-6 5G nano camera modes and transitions

Mode	Description	Next mode	Transition control
Camera App Off	The iPod performs only non-camera functions.	Preview	iPod only
Preview	The iPod screen displays the image that can be captured as video. In this mode and Recording mode, the iPod's music capabilities are disabled.	Recording	iPod or accessory
		Camera App Off	iPod only
Recording	The iPod is recording video	Preview	iPod or accessory

Mode	Description	Next mode	Transition control
		Camera App Off	iPod only

If a camera accessory is designed to receive feedback from the 5G nano, it must register for notifications, as specified in [Camera Accessory Identification](#) (page 192). After registration, the iPod notifies the accessory of its current mode and then sends notifications whenever it changes mode.

Camera Sessions

A 5G nano camera session begins when the user launches the iPod's Camera Application. The application starts in Preview mode.

When the iPod is in Preview mode the user can order it to start recording video, using either the iPod or the accessory. The iPod goes to Recording mode and records video. When the user stops recording, using either the iPod or the accessory, the iPod returns to Preview mode.

In either Preview or Recording mode the user can quit the Camera Application, using the iPod. The iPod quits the application and returns to its non-camera functionality.

The following is a typical sequence of user actions to capture a video:

1. Launch the Camera application on the iPod. This sets Preview Mode.
2. Order video recording to start, using either the iPod or the accessory.
3. Order video recording to end, using either the iPod or the accessory, or quit the Camera Application, using the iPod.

Accessory Feedback

In some situations, the user may operate the 5G nano's camera functions by means of an accessory without being able to see the iPod or otherwise get feedback from it. Thus it is desirable, but not required, for camera-control accessories to provide feedback to the user about the iPod camera's modes.

A simple feedback model might be for the accessory to provide a tristate indicator, such as a LED display, with OFF, MIDDLE, and ON states. When the accessory is connected to the iPod, and while it is going through its identification and authentication process, the indicator would be OFF. Its MIDDLE and ON states would then follow the guidelines shown in [Table 3-7](#) (page 191).

Table 3-7 Suggested accessory feedback indications

5G nano mode	Indicator	Comments
Camera App Off	OFF	Default accessory feedback state
Preview	MIDDLE	MIDDLE state indicates Preview mode
Recording	ON	Indicator remains ON until the iPod notifies the accessory that it has gone to Preview or Camera App Off mode, or communication between the accessory and the iPod is interrupted.

Camera Accessory Identification

Every accessory that supports the 5G nano camera technology described in this document must identify and authenticate itself for at least the General and Simple Remote lingoes (Lingoes 0x00 and 0x02), as specified in ["Accessory Identification"](#) (page 519).

After it has received an `AckDevAuthenticationStatus` command with a status of 0x00 from the iPod (completing the authentication process), the accessory can start sending `CameraButtonStatus` commands to it. An accessory that wants feedback from the iPod may also query the iPod's notification capabilities, using `GetSupportedEventNotification` and `RetSupportedEventNotification`, then send a `SetEventNotification` command, as described in [Camera Interface Commands](#) (page 192). To get camera feedback, the `SetEventNotification` command sets bit 4 (Camera Notifications) in its bitmask. The iPod replies with an `iPodNotification` command, passing the current mode of its Camera Application. Because the user can set the iPod's camera to either Preview or Recording mode, the accessory must be prepared to configure itself to be compatible with any of the modes listed in [Table 3-6](#) (page 190).

If it has sent a `SetEventNotification` command, the accessory must be prepared to receive `iPodNotification` commands from the iPod every time the iPod's camera state changes. The accessory may use these notifications to provide feedback to the user, as suggested in [Accessory Feedback](#) (page 191).

Camera Interface Commands

Accessory communication with the 5G nano camera is implemented by five iAP commands, as shown in [Table 3-8](#) (page 192).

Table 3-8 iPod camera interface commands

Lingo ID	Cmd ID	Cmd name	Direction	Parameters
0x00	0x4F	<code>GetSupportedEventNotification</code>	Acc to Dev	{transID:2}
0x00	0x51	<code>RetSupportedEventNotification</code>	Dev to Acc	{transID:2, eventNotifyMask:8}
0x00	0x49	<code>SetEventNotification</code>	Acc to Dev	{transID:2, eventNotifyMask:8}
0x00	0x4A	<code>iPodNotification</code>	Dev to Acc	{transID:2, notificationType:1, payload:1}
0x02	0x0E	<code>CameraButtonStatus</code>	Acc to Dev	{transID:2, buttonStatus:1}

Sample Command Sequences

This section describes two scenarios in which an accessory communicates with a 5G nano to control its camera. For further information about notification commands, see [Apple Device Event Notifications](#) (page 523).

Scenario 1: Video Control On and Off by the Accessory

A typical scenario for recording video under complete accessory control includes these actions:

1. The accessory is connected and authenticates itself. Its feedback indicator is OFF (see [Table 3-7](#) (page 191)).
2. The accessory sends `GetSupportedEventNotification` to the iPod to determine which event notifications it supports.
3. The iPod responds with `RetSupportedEventNotification`, passing a mask that reports which notifications it supports.
4. The accessory sends `SetEventNotification` to the iPod, requesting notifications for camera and flow control events.
5. The Camera Application is not yet launched, so the iPod sends the accessory an `iPodNotification` command with a payload of `Camera App Off` (see [Table 2-128](#) (page 162)).
6. Using iPod controls, the user launches the Camera application, which comes up in Preview mode.
7. The iPod sends the accessory an `iPodNotification` command with a payload of `Preview`.
8. The accessory changes its Indicator to the Middle state.
9. The user presses the camera button on the accessory.
10. The accessory sends a `CameraButtonStatus` command to the iPod, passing `0x01`.
11. The accessory sends a second “button-up” `CameraButtonStatus` command to the iPod, passing `0x00`.
12. The iPod transitions to its Recording mode and sends a corresponding notification to the accessory.
13. The accessory turns On its Indicator.
14. The iPod records video...
15. The user presses the camera button on the accessory.
16. The accessory sends a `CameraButtonStatus` command to the iPod, passing `0x01`.
17. The accessory sends a second “button-up” `CameraButtonStatus` command to the iPod, passing `0x00`. The iPod stops recording video.
18. The iPod transitions to its Preview mode and sends a corresponding notification to the accessory.
19. The accessory changes its Indicator to the Middle state.

As a result of this sequence, the accessory has started and ended video recording, during which time the accessory’s Indicator has been On. A typical sequence of commands that implements this scenario is listed in [Table 3-9](#) (page 193).

Table 3-9 Sample video control commands, accessory on and off

Step	Accessory command	iPod command	Comment
The accessory identifies itself as supporting the General and Simple Remote lingo and performs the actions necessary to pass accessory authentication, as described in "Accessory Identification" (page 519).			

Step	Accessory command	iPod command	Comment
1		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
2	GetSupportedEvent-Notification		requesting supported notifications
3		RetSupportedEvent-Notification	reporting that notifications for events 'flow control' and 'camera' are supported
4	SetEventNotification		setting notifications for events 'flow control' and 'camera'
5		iPodNotification	sending notification 'Camera Not Ready'
6		ACK	acknowledging 'Success (OK)' to command 'General Lingo::SetEventNotification'
The user launches the Camera application, using controls on the iPod.			
7		iPodNotification	sending notification 'Preview'
The user presses the Camera Action button on the accessory.			
8	CameraButtonStatus		Camera Action
9	CameraButtonStatus		button up
10		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
11		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
12		iPodNotification	sending notification 'Recording'
The user presses the Camera Action button on the accessory.			
13	CameraButtonStatus		Camera Action
14	CameraButtonStatus		button up
15		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
16		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'

Step	Accessory command	iPod command	Comment
17		iPodNotification	sending notification 'Preview'

Scenario 2: Video Control On by the iPod and Off by the Accessory

This scenario differs from Scenario 1 in that the iPod starts video recording and the accessory ends it. The following actions take place:

1. The accessory is connected and authenticates itself. Its feedback indicator is OFF (see [Table 3-7](#) (page 191)).
2. The accessory sends `GetSupportedEventNotification` to the iPod to determine which event notifications it supports.
3. The iPod responds with `RetSupportedEventNotification`, passing a mask that reports which notifications it supports.
4. The accessory sends `SetEventNotification` to the iPod, requesting notifications for camera and flow control events.
5. The Camera Application is not yet launched, so the iPod sends the accessory an `iPodNotification` command with a payload of Camera App Off (see [Table 2-128](#) (page 162)).
6. Using iPod controls, the user launches the Camera application, which starts in Preview mode.
7. The iPod sends the accessory an `iPodNotification` command with a payload of Preview.
8. The accessory changes its Indicator to the Middle state.
9. The user starts Recording using controls on the iPod.
10. The iPod sends the accessory an `iPodNotification` command with a payload of Recording.
11. The accessory turns On its Indicator.
12. The iPod records video...
13. The user presses the camera button on the accessory.
14. The accessory sends a `CameraButtonStatus` command to the iPod, passing 0x01.
15. The accessory sends a second “button-up” `CameraButtonStatus` command to the iPod, passing 0x00. The iPod stops recording video.
16. The iPod transitions to its Preview mode and sends a corresponding notification to the accessory.
17. The accessory changes its Indicator to the Middle state.

As a result of this sequence, the iPod has started and the accessory has ended a video recording, during which time the accessory's Indicator has been On. A typical sequence of commands that implements this scenario is listed in [Table 3-10](#) (page 196).

Table 3-10 Sample video control commands, iPod on and accessory off

Step	Accessory command	iPod command	Comment
The accessory identifies itself as supporting the General and Simple Remote lingo and performs the actions necessary to pass accessory authentication, as described in "Accessory Identification" (page 519).			
1		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
2	GetSupportedEvent-Notification		requesting supported notifications
3		RetSupportedEvent-Notification	reporting that notifications for events 'flow control' and 'camera' are supported
4	SetEventNotification		setting notifications for events 'flow control' and 'camera'
5		iPodNotification	sending notification 'Camera Not Ready'
6		ACK	acknowledging 'Success (OK)' to command 'General Lingo::SetEventNotification'
The user launches the Camera application, using controls on the iPod.			
7		iPodNotification	sending notification 'Preview'
The user starts recording video, using controls on the iPod.			
8		iPodNotification	sending notification 'Recording'
The user presses the Camera Action button on the accessory.			
9	CameraButtonStatus		Camera Action
10	CameraButtonStatus		button up
11		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
12		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
13		iPodNotification	sending notification 'Preview'

USB Human Interface Device Reports

Four commands in the Simple Remote lingo support the registration and exchange of USB Human Interface Device (HID) reports. These commands are listed in [Table 3-11](#) (page 197).

Table 3-11 USB HID commands

Command	ID	Direction	Parameters
0x0F	RegisterDescriptor	Acc to Dev	index:1, VID:2, PID:2, country_code:1, descriptor:<var>
0x10	SendHIDReportToiPod	Acc to Dev	index:1, report_type:1, report:<var>
0x11	SendHIDReportToAcc	Dev to Acc	index:1, report_type:1, report:<var>
0x12	UnregisterDescriptor	Acc to Dev	index:1

The contents of the HID reports that an accessory may exchange with an Apple device are determined by the *USB Device Class Definition for HID* specification. The HID reports are sent as opaque byte streams. Certain HID Consumer Page (0x0C) usages are also supported by iOS in addition to the standard HID alphanumeric keycodes. These controls are listed in [Table 3-12](#) (page 197) and are described in detail in the *USB HID Usage Tables* specification, Version 1.12.

Note: To control display brightness, accessories must use the iAP Display Remote lingo commands `GetiPodStateInfo` and `SetiPodStateInfo`.

Table 3-12 USB HID Consumer Page controls supported by iOS

HID Usage ID	HID Usage Name	iOS Function
0x0030	Power	Lock
0x0040	Menu	Home
0x00B5	Scan Next Track	Transport Right
0x00B6	Scan Previous Track	Transport Left
0x00CD	Play/Pause	Play/Pause
0x00E2	Mute	Mute
0x00E9	Volume Increment	Louder
0x00EA	Volume Decrement	Softer
0x01AE	AL Keyboard Layout	Toggle Onscreen Keyboard
0x01B1	AL Screen Saver	Picture Frame
0x0221	AC Search	Spotlight

Note: The glyphs that the accessory displays on its physical keycaps must be approved by Apple MFi Licensing.

User Interface Accessibility Commands

Eight commands in the Simple Remote lingo help accessories improve the Apple device's user interface accessibility. These commands are listed in [Table 3-13](#) (page 198).

Table 3-13 Simple Remote lingo accessibility commands

ID	Command	Direction	Parameters
0x13	AccessibilityEvent	Acc to Dev	{eventType;1, eventData;<var>}
0x14	GetAccessibilityParameter	Acc to Dev	paramType;1
0x15	RetAccessibilityParameter	Dev to Acc	{paramType;1, paramValue;<var>}
0x16	SetAccessibilityParameter	Acc to Dev	{paramType;1, paramValue;<var>}
0x17	GetCurrentItemProperty	Acc to Dev	propertyType;1
0x18	RetCurrentItemProperty	Dev to Acc	{propertyType;1, propertyValue;<var>}
0x19	SetContext	Acc to Dev	context;1
0x1A	AccParameterChanged	Dev to Acc	{paramType;1, paramValue;<var>}

Command 0x00: ContextButtonStatus

Direction: Accessory to Apple device

The accessory sends this command to the Apple device when a button event occurs. The button status is a bitmask representing each button that is currently pressed. The accessory must send the button status packet repeatedly at intervals between 30 and 100 ms, while one or more buttons are pressed. When all buttons are released, the accessory must send a button status packet with a 0x00 payload to indicate that no buttons are pressed. The Apple device does not return a packet to the accessory in response to this command.

Table 3-14 ContextButtonStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03–0x06	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo

Byte number	Value	Comment
4	0x00	Command ID: <code>ContextButtonStatus</code>
5	0xNN	Byte index 0, button states 7:0; see Table 3-15 (page 199) for a list of button states recognized by Apple devices.
6	0xNN	Byte index 1, button states 15:8 (optional)
7	0xNN	Byte index 2, button states 23:16 (optional)
8	0xNN	Byte index 3, button states 31:24 (optional)
(last byte)	0xNN	Checksum

[Table 3-15](#) (page 199) lists the available buttons and their bitmasks.

Note: iOS devices support only a subset of this list, as specified in "Controlling Applications Using the Simple Remote Lingo" (page 53).

Table 3-15 Button states

Button name	Number	Byte index	Button bitmask
Play/Pause	0	0x00	0x01
Volume Up	1	0x00	0x02
Volume Down	2	0x00	0x04
Next Track	3	0x00	0x08
Previous Track	4	0x00	0x10
Next Album	5	0x00	0x20
Previous Album	6	0x00	0x40
Stop	7	0x00	0x80
Play/Resume	8	0x01	0x01
Pause	9	0x01	0x02
Mute toggle	10	0x01	0x04
Next Chapter	11	0x01	0x08
Previous Chapter	12	0x01	0x10
Next Playlist	13	0x01	0x20
Previous Playlist	14	0x01	0x40

Button name	Number	Byte index	Button bitmask
Shuffle Setting Advance	15	0x01	0x80
Repeat Setting Advance	16	0x02	0x01
Power On	17	0x02	0x02
Power Off	18	0x02	0x04
Backlight for 30 Seconds	19	0x02	0x08
Begin Fast Forward	20	0x02	0x10
Begin Rewind	21	0x02	0x20
Menu	22	0x02	0x40
Select	23	0x02	0x80
Up Arrow	24	0x03	0x01
Down Arrow	25	0x03	0x02
Backlight Off	26	0x03	0x04
Reserved	31:27	0x03	0xF8

Notes

- The Begin Fast Forward and Begin Rewind states must be sent every 30 to 100 ms to assure continuous fast forward or rewind actions.
- If the user holds down the Fast Forward button (sending repeated Fast Forward states) through a track change, the new track begins playing at normal speed.
- The Menu button navigates only within iPod/Music/Video applications and does not return the user to the Home screen.

Command 0x01: ACK

Direction: Apple device to Accessory

The Apple device sends this command in response to any command sent from the accessory, except command 0x00. An ACK response is sent when a command that does not return any data has completed, when a bad parameter is received, or when an unsupported or invalid command is received.

Table 3-16 ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x01	Command ID: ACK
5	0xNN	Status of command received (see Table 3-17 (page 201))
6	0xNN	ID of the command being acknowledged
7	0xNN	Checksum

Table 3-17 Command status codes

Code	Command status
0x00	Command OK
0x01	Unknown track category (not applicable)
0x02	Command failed (valid command, did not succeed)
0x03	Out of resources (Apple device internal allocation failed)
0x04	Bad parameter (command or input parameters invalid)
0x05	Unknown track ID (not applicable)
0x06	Command pending (cmdPendTime parameter returned)
0x07	Not authenticated (not authenticated)
0x08	Mismatched authentication protocol version
0x09–0x15	Reserved
0x16	HID descriptor index already in use
0x17–0xFF	Reserved

Command 0x03: VideoButtonStatus

Direction: Accessory to Apple device

The accessory sends this command to the Apple device when a video-specific button event occurs. The button status is a bitmask representing each button that is currently pressed. The button status packet must be repeatedly sent by the accessory at intervals between 30 ms and 100 ms while one or more buttons are

pressed. When all buttons are released, the accessory must send a button status packet with a 0x00 payload to indicate that no buttons are pressed. In response, the Apple device will return an ACK packet containing the command status to the accessory.

Note: Apple products running iOS 3.2 support only the Stop, Play/Resume, and Pause button values.

Table 3-18 VideoButtonStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x03	Command ID: VideoButtonStatus
5	0xNN	Byte index 0, video-specific button states 7:0; see Table 3-19 (page 202) for a list of video-specific button states recognized by Apple devices.
6	0xNN	Byte index 1, button states 15:8 (optional)
7	0xNN	Byte index 2, button states 23:16 (optional)
8	0xNN	Byte index 3, button states 31:24 (optional)
(last byte)	0xNN	Checksum

Table 3-19 Video-specific button values

Button name	Number	Byte index	Button bitmask
Play/Pause	0	0x00	0x01
Next video	1	0x00	0x02
Previous video	2	0x00	0x04
Stop	3	0x00	0x08
Play/Resume	4	0x00	0x10
Pause	5	0x00	0x20
Begin FF	6	0x00	0x40
Begin REW	7	0x00	0x80
Next chapter	8	0x01	0x01
Previous chapter	9	0x01	0x02

Button name	Number	Byte index	Button bitmask
Next frame	10	0x01	0x04
Previous frame	11	0x01	0x08
Reserved	12:15	0x01	0xF0
Reserved	16:23	0x02	0xFF
Reserved	24:31	0x03	0xFF

The Next Video and Previous Video button commands skip to the next or previous video. If the current video has played less than two seconds, Previous Video backs up to the beginning of the previous video; if it has played more than two seconds, it backs up to the beginning of the current video. These commands skip to the next or previous video even when the current video has chapters. If the current video has chapters, the Next Chapter and Previous Chapter commands skip to the next or previous chapter; otherwise they have no effect. If the video has chapters and has played more than two seconds of the current chapter, the Previous Chapter command backs up to the beginning of the current chapter; if the track has played less than two seconds, it backs up to the beginning of the previous chapter.

Command 0x04: AudioButtonStatus

Direction: Accessory to Apple device

The accessory sends this command to the Apple device when an audio-specific button event occurs. The button status is a bitmask representing each button that is currently pressed. The button status packet must be repeatedly sent by the accessory at intervals between 30 ms and 100 ms while one or more buttons are pressed. When all buttons are released, the accessory must send a button status packet with a 0x00 payload to indicate that no buttons are pressed. In response, the Apple device will return to the accessory an ACK message containing the command status.

Note: Apple products running iOS 3.2 support only the Stop, Play/Resume, and Pause button values.

Table 3-20 AudioButtonStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x04	Command ID: AudioButtonStatus
5	0xNN	Byte index 0, audio-specific button states 7:0; see Table 3-21 (page 204) for a list of audio-specific button states recognized by Apple devices.

Byte number	Value	Comment
6	0xNN	Byte index 1, button states 15:8 (optional)
7	0xNN	Byte index 2, button states 23:16 (optional)
8	0xNN	Byte index 3, button states 31:24 (optional)
(last byte)	0xNN	Checksum

Table 3-21 Audio-specific button values

Button name	Number	Byte index	Button bitmask
Play/Pause	0	0x00	0x01
Volume Up	1	0x00	0x02
Volume Down	2	0x00	0x04
Next Track	3	0x00	0x08
Previous Track	4	0x00	0x10
Next Album	5	0x00	0x20
Previous Album	6	0x00	0x40
Stop	7	0x00	0x80
Play/Resume	8	0x01	0x01
Pause	9	0x01	0x02
Mute toggle	10	0x01	0x04
Next chapter	11	0x01	0x08
Previous chapter	12	0x01	0x10
Next playlist	13	0x01	0x20
Previous playlist	14	0x01	0x40
Shuffle setting advance	15	0x01	0x80
Repeat setting advance	16	0x02	0x01
Begin FF	17	0x02	0x02
Begin REW	18	0x02	0x04
Record	19	0x02	0x08
Reserved	20:23	0x02	0xF0

Button name	Number	Byte index	Button bitmask
Reserved	24:31	0x03	0xFF

The Next and Previous Track, Album, and Playlist button commands skip to the next or previous track, album, or playlist. If the current track, album, or playlist has played less than two seconds, the Previous command backs up to the beginning of the previous one; if it has played more than two seconds, it backs up to the beginning of the current one. These commands skip to the next or previous track, album, or playlist even when the current one has chapters. If the current track has chapters, the Next Chapter and Previous Chapter commands skip to the next or previous chapter; otherwise they have no effect. If the track has chapters and has played more than two seconds of the current chapter, the Previous Chapter command backs up to the beginning of the current chapter; if the track has played less than two seconds, it backs up to the beginning of the previous chapter.

Command 0x0B: iPodOutButtonStatus

Direction: Accessory to Apple device

WARNING: The Apple device must be in iPod Out mode when the accessory sends this command.

The accessory sends this command to inform the Apple device that the user has pushed one or more accessory buttons intended to control the Apple device. The `iPodOutButtonMask` field is a bitmask representing each button that is currently pressed. The accessory should repeatedly send the packet shown in [Table 3-22](#) (page 205) at an interval of 30 to 100 ms while one or more buttons are pressed (note that bytes 7-9 in the packet are optional). When all buttons are released, the accessory must send a `iPodOutButtonStatus` command with a packet length of 0x04 and a `iPodOutButtonMask` value of 0x0.

In response to each packet, the Apple device returns a Simple Remote ACK command with the command status. The button events reported by this command are handled by the Apple device in a context-specific manner.

Table 3-22 `iPodOutButtonStatus` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x02	Lingo ID: Simple Remote lingo
4	0x0B	Command ID: <code>iPodOutButtonStatus</code>
5	0xNN	<code>buttonSource</code> : Location of button; see Table 3-23 (page 206).
6	0xNN	<code>iPodOutButtonMask</code> : Button status bitmask (bits 07-00); see Table 3-24 (page 206).

Byte number	Value	Comment
7	0xNN	iPodOutButtonMask: Button status bitmask (bits 15-08): optional
8	0xNN	iPodOutButtonMask: Button status bitmask (bits 23-16): optional
9	0xNN	iPodOutButtonMask: Button status bitmask (bits 31-24): optional
last byte	0xNN	Checksum

Table 3-23 iPod Out control locations

Value	Meaning
0x00	Car center console
0x01	Steering wheel
0x02	Car dashboard
0x03-0xFF	Reserved

Table 3-24 iPod Out button status values

Bit	Meaning
00	Select event
01	Left event
02	Right event
03	Up event
04	Down event
05	Menu button event
06-07	Reserved

Command 0x0C: RotationInputStatus

Direction: Accessory to Apple device

WARNING: The Apple device must be in iPod Out mode when the accessory sends this command.

The accessory sends this command to inform the Apple device that the user is acting on a rotating device (such as a jog wheel, control knob with detents, or a free-turning wheel) intended to control the Apple device. The accessory should send command packets at intervals of no more than 100 ms while the user action is in progress, and then it must send a final packet when the user action has finished.

Note: If the accessory does not send packets at least every 100 ms, the Apple device may not respond as expected. However, the Apple device should respond properly to a late packet if it contains a correct `userActionDurationMs` value.

If the user has turned the rotating device to a stationary position and is holding it in place (assuming the accessory can detect such an action), this command must be sent with a `rotationAction` value of 0x02 at intervals of no more than 100 ms while the user is holding the input device in place. Once the user stops acting on the rotating device, the accessory must send this command with a `rotationAction` value of 0x00.

Table 3-25 RotationInputStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0F	Length of packet
3	0x02	Lingo ID: Simple Remote lingo
4	0x0C	Command ID: RotationInputStatus
5	0xNN	<code>userActionDurationMs</code> (bits 31:24): Number of milliseconds since the start of the current user action.
6	0xNN	<code>userActionDurationMs</code> (bits 23:16).
7	0xNN	<code>userActionDurationMs</code> (bits 15:8).
8	0xNN	<code>userActionDurationMs</code> (bits 7:0).
9	0xNN	<code>rotationSource</code> : Location of wheel; see Table 3-23 (page 206).
10	0xNN	<code>controllerType</code> : Type of wheel; see Table 3-26 (page 208).
11	0xNN	<code>rotationDirection</code> : Direction of rotation; see Table 3-27 (page 208).
12	0xNN	<code>rotationAction</code> : Rotation action; see Table 3-28 (page 208).
13	0xNN	<code>rotationType</code> : Type of rotation; see Table 3-29 (page 208).
14	0xNN	<code>detentsOrDegreesMoved</code> (bits 15:8): Number of detents or degrees the user has moved the wheel since the last RotationInputStatus packet
15	0xNN	<code>detentsOrDegreesMoved</code> (bits 7:0)
16	0xNN	<code>detentsOrDegreesTotal</code> (bits 15:8): Constant number of detents or degrees in a full turn of the wheel (maximum = 360)
17	0xNN	<code>detentsOrDegreesTotal</code> (bits 7:0)
18	0xNN	Checksum

Table 3-26 Wheel types

Value	Meaning
0x00	Free wheel
0x01	Jog wheel
0x02-0xFF	Reserved

Table 3-27 Rotation directions

Value	Meaning
0x00	Counterclockwise (left)
0x01	Clockwise (right)
0x02-0xFF	Reserved

Table 3-28 Rotation actions

Value	Meaning
0x00	Rotation action completed; user has released control
0x01	Rotation in progress; wheel turning
0x02	Rotation repeat; the user has not advanced the wheel from the position reported in the last <code>RotationInputStatus</code> packet
0x03-0xFF	Reserved

Table 3-29 Rotation types

Value	Meaning
0x00	Wheel has detents
0x01	Wheel reports angular degrees
0x02-0xFF	Reserved

Examples of Using `RotationInputStatus`

Following are two examples of `RotationInputStatus` packets sent by an accessory.

Example with a Detent Wheel

- The user begins to turn wheel; the accessory sends a `RotationInputStatus` command to the Apple device after 100 ms

- ❑ `userActionDurationMs = 0`
- ❑ `buttonSource = car center console (0x00)`
- ❑ `controllerType = free wheel (0x00)`
- ❑ `rotationDirection = right (0x01)`
- ❑ `rotationAction = rotation started (0x01)`
- ❑ `rotationType = detents (0x00)`
- ❑ `detentsMoved = 2 (0x02)`
- ❑ `detentsTotal = 16 (0x10)`
- The user continues to turn wheel until 100 ms after starting rotation. During that time, the accessory sends a second `RotationInputStatus` command to the Apple device
 - ❑ `userActionDurationMs = 100`
 - ❑ `buttonSource = car center console (0x00)`
 - ❑ `controllerType = free wheel (0x00)`
 - ❑ `rotationDirection = right (0x01)`
 - ❑ `rotationAction = rotation in progress (0x01)`
 - ❑ `rotationType = detents (0x00)`
 - ❑ `detentsMoved = 3 detents since last packet (0x03)`
 - ❑ `detentsTotal = 16 (0x10)`
- The user stops after 200 ms; the accessory sends a `RotationInputStatus` command to the Apple device indicating completion
 - ❑ `userActionDurationMs = 100`
 - ❑ `buttonSource = car center console (0x00)`
 - ❑ `controllerType = free wheel (0x00)`
 - ❑ `rotationDirection = right (0x01)`
 - ❑ `rotationAction = rotation action completed (0x00)`
 - ❑ `rotationType = detents (0x00)`
 - ❑ `detentsMoved = 0 detents since last packet (0x00)`
 - ❑ `detentsTotal = 16 (0x10)`

Example with a Jog Wheel

- The user begins to turn wheel; the accessory sends a `RotationInputStatus` command to the Apple device after 100 ms
 - ❑ `userActionDurationMs = 100`
 - ❑ `buttonSource = car center console (0x00)`
 - ❑ `controllerType = jog wheel (0x01)`

- ❑ rotationDirection = right (0x01)
- ❑ rotationAction = rotation started (0x01)
- ❑ rotationType = degrees (0x01)
- ❑ degreesMoved = 32 (0x20)
- ❑ degreesTotal = 360 (0x168)
- The user does not move the wheel from the 32 degree position during the 100-295 ms interval before releasing the wheel; the accessory sends a `RotationInputStatus` command to the Apple device
 - ❑ userActionDurationMs = 100
 - ❑ buttonSource = car center console (0x00)
 - ❑ controllerType = jog wheel (0x01)
 - ❑ rotationDirection = right (0x01)
 - ❑ rotationAction = rotation repeat (0x02)
 - ❑ rotationType = degrees (0x01)
 - ❑ degreesMoved = 0 (0x00)
 - ❑ degreesTotal = 360 (0x168)
- The user releases the jog wheel after 295 ms. The accessory sends a `RotationInputStatus` command to the Apple device indicating completion
 - ❑ userActionDurationMs = 95
 - ❑ buttonSource = car center console (0x00)
 - ❑ controllerType = jog wheel (0x01)
 - ❑ rotationDirection = right (0x01)
 - ❑ rotationAction = rotation action completed (0x00)
 - ❑ rotationType = degrees (0x01)
 - ❑ degreesMoved = 0 (0x00)
 - ❑ degreesTotal = 360 (0x168)

Command 0x0D: RadioButtonStatus

Direction: Accessory to Apple device

The accessory sends this command to a 5G nano iPod to initiate a tagging action by the iPod's Radio Application. Once the accessory has authenticated itself for the Simple Remote lingo, it may send this command at any time.

Currently the only payloads for this command are the values 0x02 and 0x00 in `buttonStatus` (byte 7). Any other values will result in the iPod returning an error value of 0x04 (Bad Parameter). The accessory should repeatedly send this command with a payload of 0x02 at intervals of 30-100 ms while its Radio Tag button is held down. When the button is released it must send a payload of 0x00 to indicate that no buttons are down. The iPod responds with an `ACK` command passing success (0x00).

This command is intended to be used with the `GetSupportedEventNotification` and `SetEventNotification` commands; see [Apple Device Event Notifications](#) (page 523).

Table 3-30 `RadioButtonStatus` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x0D	Command ID: <code>RadioButtonStatus</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command.
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>buttonStatus</code> : 0x02 = Radio button pushed to tag current song; 0x00 = button released.
8	0xNN	Checksum

Command 0x0E: `CameraButtonStatus`

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to initiate changes in the Apple device's Camera mode. Once the accessory has authenticated itself for the Simple Remote lingo, it may send this command at any time.

The accessory must send `CameraButtonStatus` commands in pairs: the first command must pass 0x01 in `buttonStatus` (byte 7) and the second command must pass 0x00. If the accessory doesn't send a second command, the Apple device assumes such a return after a 200 ms timeout. Each `CameraButtonStatus` command may pass only the value 0x00 or 0x01; otherwise, the Apple device will acknowledge it with a Bad Parameter error status. The Apple device will acknowledge each valid `CameraButtonStatus` command with a General lingo `ACK` command reporting Status OK (0x00).

This command is intended to be used with the `GetSupportedEventNotification` and `SetEventNotification` commands; see [Apple Device Event Notifications](#) (page 523).

Table 3-31 CameraButtonStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x0E	Command ID: CameraButtonStatus
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	buttonStatus: (0x01 = down, 0x00 = up)
8	0xNN	Checksum

Command 0x0F: RegisterDescriptor

Direction: Accessory to Apple device

The accessory sends this command to register a USB Human Interface Device (HID) Report Descriptor for subsequent use by the `SendHIDReportToiPod` and `SendHIDReportToAcc` commands. The Apple device creates and registers a new USB device in the iPod HID layer without parsing the descriptor. The Apple device responds to this command by sending a Simple Remote ACK command.

For a description of the process of sending HID reports via iAP, see "[USB Human Interface Device Reports](#)" (page 197).

Table 3-32 RegisterDescriptor packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x0F	Command ID: RegisterDescriptor
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	index: USB descriptor index

Byte number	Value	Comment
8	0xNN	VID [bits 15:8]: Vendor ID of the accessory; see Note below.
9	0xNN	VID [bits 7:0]
10	0xNN	PID [bits 15:8]: Product ID of the accessory; see Note below.
11	0xNN	PID [bits 7:0]
12	0xNN	country_code: Country code of the accessory; see Table 3-33 (page 213). Localized devices, such as keyboards, should pass a country code. Nonlocalized devices must pass 0x00.
13–NN	0xNN	USB HID descriptor
NN+1	0xNN	Checksum

Note: The Vendor ID and Product ID parameters of the `RegisterDescriptor` command must comply with the VID/PID requirements specified in “Using an Apple Device as a USB Host” in *MFi Accessory Hardware Specification*.

Table 3-33 Country codes

Decimal	Hexadecimal	Country or localization
0	00	not supported / undefined
1	01	Arabic
2	02	Belgian
3	03	Canadian-Bilingual
4	04	Canadian-French
5	05	Czech Republic
6	06	Danish
7	07	Finnish
8	08	French
9	09	German
10	0A	Greek
11	0B	Hebrew
12	0C	Hungarian
13	0D	International (ISO)

Decimal	Hexadecimal	Country or localization
14	0E	Italian
15	0F	Japan (Katakana)
16	10	Korean
17	11	Latin American
18	12	Netherlands/Dutch
19	13	Norwegian
20	14	Persian (Farsi)
21	15	Poland
22	16	Portuguese
23	17	Russian
24	18	Slovakia
25	19	Spanish
26	1A	Swedish
27	1B	Swiss/French
28	1C	Swiss/German
29	1D	Switzerland
30	1E	Taiwan
31	1F	Turkish-Q
32	20	UK
33	21	US
34	22	Croatian
35	23	Turkish-F
36–249	24–F9	reserved
250	FA	Thai
251	FB	Flemish
252	FC	Romanian
253	FD	Bulgarian

Decimal	Hexadecimal	Country or localization
254	FE	Chinese (Simplified)

Command 0x10: SendHIDReportToiPod

Direction: Accessory to Apple device

The accessory sends this command to report USB Human Interface Device (HID) events to the Apple device, such as changes in its user controls. The Apple device routes the report to the iPod HID layer without parsing it. The Apple device responds to this command by sending a Simple Remote ACK command.

Table 3-34 SendHIDReportToiPod packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x10	Command ID: SendHIDReportToiPod
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	index: USB descriptor index
8	0x00	report_type: Report type = input.
9–NN	0xNN	USB HID report
NN+1	0xNN	Checksum

Command 0x11: SendHIDReportToAcc

Direction: Apple device to Accessory

The Apple device sends this command to report USB Human Interface Device (HID) events to accessory, such as requests to change its LED display. The Apple device forwards outbound reports from its HID layer to the accessory without parsing them. The accessory must respond to this command by sending the Apple device a Simple Remote DevACK command.

Table 3-35 SendHIDReportToAcc packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x11	Command ID: SendHIDReportToAcc
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	index: USB descriptor index
8	0x01	report_type: Report type = output.
9–NN	0xNN	USB HID report
NN+1	0xNN	Checksum

Command 0x12: UnregisterDescriptor

Direction: Accessory to Apple device

The accessory sends this command to inform the Apple device that the USB HID report descriptor created by an earlier RegisterDescriptor command is no longer needed and may be discarded. The Apple device responds to this command by sending a Simple Remote ACK command.

Table 3-36 UnregisterDescriptor packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x12	Command ID: UnregisterDescriptor
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	index: USB descriptor index

Byte number	Value	Comment
8	0xNN	Checksum

Command 0x13: AccessibilityEvent

Direction: Accessory to Apple device

The accessory sends this command to notify the Apple device that the user has generated an interface event. The `eventType` parameter identifies the type of the event; based on this type, there may be data in the `eventData` parameter.

Table 3-37 AccessibilityEvent packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x13	Command ID: AccessibilityEvent
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	eventType: see Table 3-38 (page 217).
8-NN	0xNN	eventData: see Table 3-38 (page 217).
Last byte	0xNN	Checksum

Table 3-38 AccessibilityEvent events and data

eventType	Event	eventData
0x00	Move to a point on the screen. This event assumes a screen size of 65536 by 65536, with both x and y values between 0 and 65535. Point (0,0) is the top left corner of the view, regardless of the orientation of the Apple device.	Byte 8: X coordinate (bits 15:8) Byte 9: X coordinate (bits 7:0) Byte 10: Y coordinate (bits 15:8) Byte 11: Y coordinate (bits 7:0)
0x01	Move To First	No event data
0x02	Move To Last	No event data
0x03	Move To Next	No event data

eventType	Event	eventData
0x04	Move To Previous	No event data
0x05	Scroll Left Page	No event data
0x06	Scroll Right Page	No event data
0x07	Scroll Up Page	No event data
0x08	Scroll Down Page	No event data
0x09	Scroll to a point on the screen. This event assumes a screen size of 65536 by 65536, with both x and y values between 0 and 65535. Point (0,0) is the top left corner of the view, regardless of the orientation of the Apple device.	Byte 8: X coordinate (bits 15:8) Byte 9: X coordinate (bits 7:0) Byte 10: Y coordinate (bits 15:8) Byte 11: Y coordinate (bits 7:0)
0x0A	Send text from the accessory to the input field of the Apple device. Text may be sent in multiple sections, each preceded by its section index and the index of the last section of text. If the text fits into one command, both of these numbers must be 0x00. Multiple text sections must be sent as consecutive <code>AccessibilityEvent</code> commands with the same transaction ID. The whole text, with all its sections concatenated, must be a single null-terminated UTF-8 string.	Byte 8: current section index (bits 15:8) Byte 9: current section index (bits 7:0) Byte 10: last section index (bits 15:8) Byte 11: last section index (bits 7:0) Bytes 12-NN: UTF-8 characters
0x0B	Cut	No event data
0x0C	Copy	No event data
0x0D	Paste	No event data
0x0E	Home	No event data
0x0F	Create a touch event at a point on the screen. This event assumes a screen size of 65536 by 65536, with both x and y values between 0 and 65535. Point (0,0) is the top left corner of the view, regardless of the orientation of the Apple device.	Byte 8: X coordinate (bits 15:8) Byte 9: X coordinate (bits 7:0) Byte 10: Y coordinate (bits 15:8) Byte 11: Y coordinate (bits 7:0) Byte 12: Touch event type: 0x00: Began 0x01: Moved 0x02: Stationary 0x03: Ended 0x04: Canceled 0x05-0xFF: Reserved

eventType	Event	eventData
0x10	Set scale display factor (magnification)	A 16-bit unsigned integer: Byte 8: Scale factor (bits 15:8) Byte 9: Scale factor (bits 7:0)
0x11	Center display around a point on the screen. This event assumes a screen size of 65536 by 65536, with both x and y values between 0 and 65535. Point (0,0) is the top left corner of the view, regardless of the orientation of the Apple device.	Byte 8: X coordinate (bits 15:8) Byte 9: X coordinate (bits 7:0) Byte 10: Y coordinate (bits 15:8) Byte 11: Y coordinate (bits 7:0)
0x12	Pause speaking	No event data
0x13	Resume speaking	No event data
0x14	Read all text from current point	No event data
0x15	Read all text from top	No event data
0x16	Send text from the accessory to an iPhone, to be spoken to the user by the iPhone's text-to-speech engine. Text may be sent in multiple sections, each preceded by its section index and the index of the last section of text. If the text fits into one command, both of these numbers must be 0x00. Multiple text sections must be sent as consecutive <code>AccessibilityEvent</code> commands with the same transaction ID. The whole text to be spoken, with all its sections concatenated, must be a single null-terminated UTF-8 string.	Byte 8: current section index (bits 15:8) Byte 9: current section index (bits 7:0) Byte 10: last section index (bits 15:8) Byte 11: last section index (bits 7:0) Bytes 12-NN: UTF-8 characters
0x17-0xFF	Reserved	

Command 0x14: GetAccessibilityParameter

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to get the value of one of the Apple device's accessibility parameters.

Table 3-39 GetAccessibilityParameter packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload

Byte number	Value	Comment
3	0x02	Lingo ID: Simple Remote lingo
4	0x14	Command ID: GetAccessibilityParameter
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	paramType: The type of parameter to be returned; see Table 3-41 (page 220).
8	0xNN	Checksum

Command 0x15: RetAccessibilityParameter

Direction: Apple device to Accessory

The Apple device sends this command to the accessory, in response to a GetAccessibilityParameter command, to return the value of one of the Apple device's accessibility parameters.

Table 3-40 RetAccessibilityParameter packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x15	Command ID: RetAccessibilityParameter
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	paramType: the parameter type requested by GetAccessibilityParameter; see Table 3-41 (page 220).
8-NN	0xNN	paramValue: the value of the requested parameter; see Table 3-41 (page 220).
Last byte	0xNN	Checksum

Table 3-41 Accessibility parameter types and values

Type ID	Parameter	Value	Notes
0x00	VoiceOver volume	An 8-bit unsigned integer between 0 (mute) and 255 (full volume).	

Type ID	Parameter	Value	Notes
0x01	Speaking rate	An 8-bit unsigned integer between 0 (slowest) and 255 (fastest). 127 is normal.	
0x02	VoiceOver enabled	An 8 bit unsigned integer that encodes a Boolean. Zero = false, nonzero = true.	Cannot be set by <code>SetAccessibilityParameter</code> ; use <code>SetiPodPreferences</code> . See Table 2-66 (page 120), Class 0x14.
0x03-0xFF	Reserved		

Command 0x16: SetAccessibilityParameter

Direction: Accessory to Apple device

The accessory sends this command to set the value of one of the Apple device's accessibility parameters.

Table 3-42 SetAccessibilityParameter packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x16	Command ID: SetAccessibilityParameter
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	paramType: the accessibility parameter type to be set; see Table 3-41 (page 220). Type 0x02, VoiceOver enabled, cannot be set.
8-NN	0xNN	paramValue: the value of the parameter to be set; see Table 3-41 (page 220).
Last byte	0xNN	Checksum

Command 0x17: GetCurrentItemProperty

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to get a specific property of the item currently selected in the Apple device's user interface.

Note: This command may cause `RetCurrentItemProperty` to return large amounts of data. If necessary, the accessory may cancel the whole transaction by passing the ID of `GetCurrentItemProperty` to the General lingo `Cancel` command (0x50).

Table 3-43 `GetCurrentItemProperty` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x17	Command ID: <code>GetCurrentItemProperty</code>
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	propertyType: The type of property to be returned; see Table 3-45 (page 223).
8	0xNN	Checksum

Command 0x18: `RetCurrentItemProperty`

Direction: Apple device to Accessory

The Apple device sends this command to the accessory, in response to a `GetCurrentItemProperty` command or asynchronously, to return the value of one of the properties of the Apple device's currently selected user interface item.

Table 3-44 `RetCurrentItemProperty` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x18	Command ID: <code>RetCurrentItemProperty</code>
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]

Byte number	Value	Comment
7	0xNN	propertyType: the parameter type requested by <code>GetAccessibilityParameter</code> ; see Table 3-45 (page 223).
8-NN	0xNN	propertyValue: the value of the requested parameter; see Table 3-45 (page 223).
Last byte	0xNN	Checksum

Table 3-45 Currently selected item property types and values

ID	Property	Value
0x00	Label	A null-terminated UTF-8 string. The string may be sent in multiple sections, each preceded by its section index and the index of the last section of text. If the whole string fits into one command, both of these numbers must be 0x00. Multiple sections of the string must be sent as consecutive <code>RetCurrentItemProperty</code> commands with the same transaction ID.
0x01	Value	
0x02	Hint	
		Byte 8: current section index (bits 15:8) Byte 9: current section index (bits 7:0) Byte 10: last section index (bits 15:8) Byte 11: last section index (bits 7:0) Bytes 12-NN: UTF-8 characters
0x03	Frame	The enclosing rectangle of the currently selected item. The graphics environment assumes a screen size of 65536 by 65536, with both x and y values between 0 and 65535. Point (0,0) is the top left corner of the view, regardless of the orientation of the Apple device. Byte 8: top left X coordinate (bits 15:8) Byte 9: top left X coordinate (bits 7:0) Byte 10: top left Y coordinate (bits 15:8) Byte 11: top left Y coordinate (bits 7:0) Byte 12: bottom right X coordinate (bits 15:8) Byte 13: bottom right X coordinate (bits 7:0) Byte 14: bottom right Y coordinate (bits 15:8) Byte 15: bottom right Y coordinate (bits 7:0)

ID	Property	Value
0x04	Traits	<p>Traits are attributes of the selected item. Zero or more traits may be returned as a list of 16-bit big endian values. The number of traits returned equals the length of the packet minus 3 and divided by 2.</p> <p>Currently defined traits and their values are the following:</p> <ul style="list-style-type: none"> Button: 0x0000 Link: 0x0001 Search Field: 0x0002 Image: 0x0003 Selected: 0x0004 Sound: 0x0005 Keyboard Key: 0x0006 Static Text: 0x0007 Summary Element: 0x0008 Not Enabled: 0x0009 Updates Frequently: 0x000A Starts Media Session: 0x000B Adjustable: 0x000C Back Button: 0x000D Map: 0x000E Delete Key: 0x000F

Command 0x19: SetContext

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to set the Apple device's current user interface context.

Table 3-46 SetContext packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x19	Command ID: SetContext
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]

Byte number	Value	Comment
7	0xNN	paramType: The user interface context to be set; see Table 3-47 (page 225).
8	0xNN	Checksum

Table 3-47 User interface contexts

ID	Context	Description
0x00	None	There is no current context.
0x01	Header	The user is working in a header.
0x02	Link	The user is executing a link.
0x03	Form	The user is filling out a form.
0x04	Cursor	The user is manipulating the cursor.
0x05-0xFF	Reserved	

Command 0x1A: AccParameterChanged

Direction: Apple device to Accessory

The Apple device sends this command to the accessory whenever its VoiceOver feature is enabled or disabled, either because the accessory changed the Accessibility Preference or because the user changed this setting on the Apple device. Note that user changes may also be done through another accessibility-capable accessory.

This notification is sent to any connected accessory that has set its accessibility capability bit (0x11) when authenticating via IDPS. Commands sent to the accessory after it has received a notification that VoiceOver is no longer enabled, but before it has received a notification that VoiceOver is enabled, may be ignored.

Table 3-48 AccParameterChanged packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x1A	Command ID: AccParameterChanged
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]

Byte number	Value	Comment
7	0xNN	paramType: the parameter type that has changed; see Table 3-41 (page 220).
8-NN	0xNN	paramValue: the new value of the changed parameter; see Table 3-41 (page 220).
Last byte	0xNN	Checksum

Command 0x81: DevACK

Direction: Accessory to Apple device

The accessory sends this command in response to a command received from the Apple device.

Table 3-49 DevACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x81	Command ID: DevACK
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	cmdStatus: 0x00 if the accessory handled the Apple device command successfully.
8	0xNN	cmdOrig: ID of the command being acknowledged
7	0xNN	Checksum

Lingo 0x03: Display Remote Lingo

The Display Remote lingo is for accessory devices that need to control the state of the Apple device, display information about the state of the Apple device on a remote display, or control the state of the Apple device equalizer. The Display Remote protocol can be used by simple inline-display remotes (remotes that have single-line display and play control buttons) and more complex devices that have full multiline graphical displays to show information about the track, artist, or album; current play or pause state; track position; battery; shuffle and time.

For example, the Display Remote protocol can be used in an automotive application to show currently playing track information on the in-vehicle display while allowing the user to browse the database stored in the Apple device. Such an application could let the user choose where to browse the database (on the in-vehicle display or on the Apple device) and switch between Extended Interface and Display Remote modes, depending on the setting. For a sample command sequence that declares this lingo, see [Table 3-311](#) (page 404).

By supporting multiple lingoes, an accessory can use the Display Remote lingo in combination with other lingoes to create a fully functional device/accessory system. Accessories can also use this lingo to control the state of the Apple device equalizer. The Display Remote lingo supports serial accessories attached to the 30-pin connector.

The Display Remote command set uses a single byte command format similar to the General and Simple Remote lingoes. Devices using the Display Remote lingo can identify using the General lingo (0x0) command `IdentifyDeviceLingoes` (0x13). See ["Command 0x13: IdentifyDeviceLingoes"](#) (page 96) for more information.

Note: The Display Remote lingo is an authenticated lingo, with some exceptions. USB accessories must authenticate before they can use the Display Remote lingo. Serial accessories have access only to the equalizer control, battery state, and sound check state commands (commands 0x01–0x07 and 0x1A–0x1E) if they do not authenticate. Please refer to [Table 3-50](#) (page 227) to determine which commands require authentication.

The Display Remote lingo can operate in notification (interrupt) mode, where the Apple device sends event notifications to the accessory, or in polled (non-interrupt) mode. In polled mode, the accessory should send requests for state change information to the Apple device.

The Display Remote commands export text as UTF-8 characters. Graphics cannot be exported. Note that Chapter count information can be retrieved only from the currently playing track.

Table 3-50 Display Remote lingo command summary

Command	ID	Data length	Protocol version	Requires authentication over serial link
ACK	0x00	0x02	1.00	No
GetCurrentEQProfileIndex	0x01	0x00	1.00	No
RetCurrentEQProfileIndex	0x02	0x04	1.00	No
SetCurrentEQProfileIndex	0x03	0x05	1.00	No
GetNumEQProfiles	0x04	0x00	1.00	No
RetNumEQProfiles	0x05	0x04	1.00	No
GetIndexedEQProfileName	0x06	0x04	1.00	No
RetIndexedEQProfileName	0x07	0xNN	1.00	No
SetRemoteEventNotification	0x08	0x04	1.02	Yes

Command	ID	Data length	Protocol version	Requires authentication over serial link
RemoteEventNotification	0x09	0xNN	1.02	Yes
GetRemoteEventStatus	0x0A	0x00	1.02	Yes
RetRemoteEventStatus	0x0B	0x04	1.02	Yes
GetiPodStateInfo	0x0C	0x01	1.02	Yes
RetiPodStateInfo	0x0D	0xNN	1.02	Yes
SetiPodStateInfo	0x0E	0xNN	1.02	Yes
GetPlayStatus	0x0F	0x00	1.02	Yes
RetPlayStatus	0x10	0x0D	1.02	Yes
SetCurrentPlayingTrack	0x11	0x04	1.02	Yes
GetIndexedPlayingTrackInfo	0x12	0x07	1.02	Yes
RetIndexedPlayingTrackInfo	0x13	0xNN	1.02	Yes
GetNumPlayingTracks	0x14	0x00	1.02	Yes
RetNumPlayingTracks	0x15	0x04	1.02	Yes
GetArtworkFormats	0x16	0x02	1.04	Yes
RetArtworkFormats	0x17	0xNN	1.04	Yes
GetTrackArtworkData	0x18	0x0C	1.04	Yes
RetTrackArtworkData	0x19	0xNN	1.04	Yes
GetPowerBatteryState	0x1A	0x00	1.02	No
RetPowerBatteryState	0x1B	0x02	1.02	No
GetSoundCheckState	0x1C	0x00	1.02	No
RetSoundCheckState	0x1D	0x01	1.02	No
SetSoundCheckState	0x1E	0x02	1.02	No
GetTrackArtworkTimes	0x1F	0x0C	1.04	Yes
RetTrackArtworkTimes	0x20	0xNN	1.04	Yes
CreateGeniusPlaylist	0x21	0x0C	1.05	Yes
IsGeniusAvailableForTrack	0x22	0x08	1.05	Yes

Command	ID	Data length	Protocol version	Requires authentication over serial link
Reserved	0x23–0xFF	N/A	N/A	N/A

Command History of the Display Remote lingo

Table 3-51 (page 229) shows the history of command changes in the Display Remote lingo:

Table 3-51 Display Remote lingo command history

Lingo version	Command changes	Features
1.00	Add: 0x00–0x07	Apple device Equalizer Setting save/restore control
1.01	None	BugFix: Equalizer state not restored on extended interface exit
1.02	Add: 0x08–0x15, 0x1A–0x1E	Event notifications, Apple device state info, playback track info, sound check, power/battery support
1.03	None	BugFix: Fix intermittent UI hang when restoring on exit
1.04	Add: 0x16–0x19, 0x1F–0x20	Track artwork, lyrics, track time position in seconds
1.05	Add: 0x21–0x22	Video browsing support added to playback commands. Chapter information can be retrieved for all tracks in the Now Playing list, not just for the currently playing track. Added support for Genius playlists.

Transferring Album Art

The Display Remote lingo includes several commands that support the transfer of album artwork from an Apple device to an accessory:

- GetArtworkFormats
- RetArtworkFormats
- GetTrackArtworkTimes
- RetTrackArtworkTimes
- GetTrackArtworkData
- RetTrackArtworkData

All album art image encoding is RGB-565, which can be transferred in both big- and small-endian formats. Artwork retrieval takes place in the following steps:

1. Retrieve the number of formats available for artwork on an Apple device using `GetArtworkFormats`. It is not necessary to call `GetArtworkFormats` more than once per session; these values will be static while the accessory is attached to the Apple device. However, there are no guarantees about the number of formats and which ones are available on a particular model or firmware version. Each `formatID` in `RetArtworkFormats` specifies both a pixel encoding, such as RGB-565 little-endian, and the image dimensions. All formats are fixed-size.
2. When the accessory wants to retrieve the artwork for a given track, it calls `GetIndexedPlayingTrackInfo` with an `infoType` of `0x08`. This returns the count of artwork available for each `formatID` associated with the track. It is possible that a track may not have artwork for a particular `formatID` or that the number of images will vary by `formatID`. A given size of album artwork may be available for only one track in the database.
3. To retrieve the list of images associated with a given track and `formatID`, the accessory calls `GetTrackArtworkTimes`. This command tells the Apple device to return the associated timestamp for each artwork. The timestamp indicates when the artwork should be displayed, expressed in milliseconds from the start of playback.
4. When the accessory wants to retrieve an individual piece of artwork, it sends `GetTrackArtworkData` to the Apple device. This requires the accessory to specify a track, a `formatID`, and the timestamp of the desired image. The Apple device returns the specified artwork and the accessory can display it whenever it chooses.

Command 0x00: ACK

Direction: Apple device to Accessory

The Apple device sends this command to acknowledge the receipt of a command from the accessory and return the command status. The command ID field indicates the accessory command for which the response is being sent. The command status indicates the result of the command (success or failure).

Table 3-52 ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x00	Command ID: ACK
5	0xNN	Command result status; see Table 3-53 (page 231).
6	0xNN	The ID for the command being acknowledged.
7	0xNN	Checksum

[Table 3-53](#) (page 231) lists the possible result values for the command result status field.

Table 3-53 Command result values

Result	Meaning
0x00	Success (OK)
0x01	Reserved
0x02	ERROR: Command failed
0x03	ERROR: Out of resources
0x04	ERROR: Bad parameter
0x05	ERROR: Unknown ID
0x06	Reserved
0x07	ERROR: Accessory not authenticated
0x08–0x11	Reserved
0x12	ERROR: Selection not genius
0x13–0xFF	Reserved

Command 0x01: GetCurrentEQProfileIndex

Direction: Accessory to Apple device

The accessory requests the current Equalizer Profile setting index. In response, the Apple device sends the "Command 0x02: RetCurrentEQProfileIndex" (page 231) packet.

Table 3-54 GetCurrentEQProfileIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x01	Command ID: GetCurrentEQProfileIndex
5	0xFA	Checksum

Command 0x02: RetCurrentEQProfileIndex

Direction: Apple device to Accessory

The Apple device sends this command, returning the current Equalizer Profile setting index, in response to the "[Command 0x01: GetCurrentEQProfileIndex](#)" (page 231) packet sent by the accessory. An Equalizer Index of 0x0 indicates that the equalizer is disabled.

Table 3-55 RetCurrentEQProfileIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x02	Command ID: RetCurrentEQProfileIndex
5	0xNN	Current Equalizer Index (bits 31:24)
6	0xNN	Current Equalizer Index (bits 23:16)
7	0xNN	Current Equalizer Index (bits 15:8)
8	0xNN	Current Equalizer Index (bits 7:0)
9	0xNN	Checksum

Command 0x03: SetCurrentEQProfileIndex

Direction: Accessory to Apple device

The accessory sets the current Equalizer Profile setting index and optionally restores the original Equalizer Setting on accessory detach. The valid Equalizer Index range can be determined by sending "[Command 0x04: GetNumEQProfiles](#)" (page 233). An Equalizer Index of 0x0 tells the Apple device that the equalizer should be disabled; a valid nonzero index enables the equalizer.

In response to this command, the Apple device returns an ACK packet with the status of this command.

Table 3-56 SetCurrentEQProfileIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x03	Command ID: SetCurrentEQProfileIndex
5	0xNN	Current Equalizer Index (bits 31:24)

Byte number	Value	Comment
6	0xNN	Current Equalizer Index (bits 23:16)
7	0xNN	Current Equalizer Index (bits 15:8)
8	0xNN	Current Equalizer Index (bits 7:0)
9	0xNN	bRestoreOnExit. Specifies whether to restore the previous Equalizer Setting on accessory exit or detach. See the discussion below.
10	0xNN	Checksum

The bRestoreOnExit Boolean byte flag determines the behavior of the Apple device when the accessory is detached from the connector. A value of 0x0 (false) indicates that the original Equalizer Setting should be discarded. A nonzero (true) value indicates that the previous Equalizer Setting should be restored when the accessory is detached from the Apple device. Anytime the SetCurrentEQProfileIndex command is sent with bRestoreOnExit equal to false, the previous equalizer state is erased and lost. If

SetCurrentEQProfileIndex is sent with bRestoreOnExit equal to true every time, the first SetCurrentEQProfileIndex command saves the original equalizer state; subsequent commands do not change the original saved equalizer state. On accessory detach, the original saved equalizer state is restored.

Command 0x04: GetNumEQProfiles

Direction: Accessory to Apple device

The accessory requests the number of Apple device Equalizer Profile settings. In response, the Apple device sends the "Command 0x05: RetNumEQProfiles" (page 233) packet.

Table 3-57 GetNumEQProfiles packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x04	Command ID: GetNumEQProfiles
5	0xF7	Checksum

Command 0x05: RetNumEQProfiles

Direction: Apple device to Accessory

The Apple device returns the number of Equalizer Profiles in it. It sends this command in response to the "Command 0x04: GetNumEQProfiles" (page 233) packet sent by the accessory. The valid profile index range for Apple device equalizer commands accepting a profile index is 0x0 to profileCount-1.

Table 3-58 RetNumEQProfiles packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x05	Command ID: RetNumEQProfiles
5	0xNN	profileCount. The Equalizer Profile count (bits 31:24).
6	0xNN	Equalizer profile count (bits 23:16)
7	0xNN	Equalizer profile count (bits 15:8)
8	0xNN	Equalizer profile count (bits 7:0)
9	0xNN	Checksum

Command 0x06: GetIndexedEQProfileName

Direction: Accessory to Apple device

The accessory requests the Apple device Equalizer Profile setting name for a given Equalizer Profile index. In response, the Apple device sends the "Command 0x07: RetIndexedEQProfileName" (page 235) packet. The valid profile index range can be obtained by sending "Command 0x04: GetNumEQProfiles" (page 233).

Table 3-59 GetIndexedEQProfileName packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x06	Command ID: GetIndexedEQProfileName
5	0xNN	Equalizer profile index (bits 31:24)
6	0xNN	Equalizer profile index (bits 23:16)

Byte number	Value	Comment
7	0xNN	Equalizer profile index (bits 15:8)
8	0xNN	Equalizer profile index (bits 7:0)
9	0xNN	Checksum

Command 0x07: RetIndexedEQProfileName

Direction: Apple device to Accessory

The Apple device returns its Equalizer Profile setting name for the specified Equalizer Profile index in response to "Command 0x06: GetIndexedEQProfileName" (page 234). The Equalizer Profile name is returned as a variable-length, null-terminated UTF-8 character array.

Note: The UTF-8 Equalizer Profile name string is not limited to 255 characters. It may be sent in either small or large packet format. The following table shows the small packet format.

Table 3-60 RetIndexedEQProfileName packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x07	Command ID: RetIndexedEQProfileName
5 ... N	0xNN...	The Equalizer Profile name, as a null-terminated UTF-8 character array.
(last byte)	0xNN	Checksum

Command 0x08: SetRemoteEventNotification

Direction: Accessory to Apple device

The accessory requests that the Apple device enable asynchronous remote event notification for specific Apple device events. Notification for each event can be enabled by setting the associated bit in the remote event bitmask (`remEventMask`). By default, all event notifications are disabled and must be explicitly enabled using this command. In response, the Apple device sends an ACK command indicating the command completion status. A remote event bitmask of 0x0 disables all remote event status notifications. On accessory detach, event notification is reset to the default disabled state.

Table 3-61 SetRemoteEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x08	Command ID: SetRemoteEventNotification
5	0xNN	remEventMask (bits 31:24). See Table 3-62 (page 236) for a list of the events for which you can enable notification.
6	0xNN	remEventMask (bits 23:16)
7	0xNN	remEventMask (bits 15:8)
8	0xNN	remEventMask (bits 7:0)
9	0xNN	Checksum

Enable notifications for the events listed in [Table 3-62](#) (page 236) by setting the bit for each event in the remote event bitmask. A value of 1 enables the notification of the Apple device state change for that event and a value of 0 disables the notification.

Table 3-62 Apple device events

Bit number	Remote Event
0	Track time position in milliseconds
1	Track playback index
2	Chapter index
3	Play status (play, pause, stop, FF, and RW)
4	Mute/UI Volume (see Notes below)
5	Power/battery
6	Equalizer setting
7	Shuffle setting
8	Repeat setting
9	Date and time setting
10	Alarm setting Deprecated; do not use

Bit number	Remote Event
11	Backlight level
12	Hold switch state
13	Sound check state
14	Audiobook speed
15	Track time position in seconds
16	Mute/UI/Absolute Volume (see Notes below)
17	Track capabilities
18	Playback engine contents
31:19	Reserved

Notes to table: When an accessory has identified itself using IDPS, it must do the following before setting bit 4 or bit 16 in the remote event bitmask of `RemoteEventNotification` to obtain notifications of volume change events:

- It must set bit 00 and bit 11 in its `AccCapsToken` value to enable analog line out and to check its volume; see [Table 2-80](#) (page 133).
- It must register a `SetiPodPreferenceToken` of Class 0x03 and value 0x01 to enable line out; see [Table 2-66](#) (page 120).

An accessory should set bit 4 or bit 16 only if it can respond appropriately to the corresponding notifications.

Command 0x09: RemoteEventNotification

Direction: Apple device to Accessory

The Apple device sends this command asynchronously whenever an enabled event change has occurred. Use "[Command 0x08: SetRemoteEventNotification](#)" (page 235) to control which events are enabled. The notification packet formats are described in more detail below. Notifications for enabled events are sent every 500 ms, with the exception of volume change notifications, which are sent every 100 ms.

Note: Notifications are sent only when the Apple device is in Power On mode; none are sent when the Apple device is in Sleep or Hibernate mode.

This is the command packet for the `RemoteEventNotification` command. See [Table 3-64](#) (page 238) to interpret the `eventNum` and `eventData` fields.

Table 3-63 RemoteEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x09	Command ID: RemoteEventNotification
5	0xNN	eventNum. A number indicating the type of event.
6 ... N	0xNN	eventData. Additional information about the event. This field is variable length; see Table 3-64 (page 238).
(last byte)	0xNN	Checksum

[Table 3-64](#) (page 238) shows the expected payload and length for each type of event notification. For example, if an accessory receives an event notification for event 0x02 (Chapter Info) then the payload is 8 bytes long and contains the track index (bytes 0–3), the chapter count (bytes 4 and 5) and the chapter index (bytes 6 and 7). If bytes 4 and 5 are all zeros, the currently playing track does not have chapters.

Table 3-64 Event notification data

Event number	Event	Event data	Data length in bytes
0x00	Track position	The new track position, in milliseconds. <ul style="list-style-type: none"> ■ Byte 0: Track Position (bits 31:24) ■ Byte 1: Track Position (bits 23:16) ■ Byte 2: Track Position (bits 15:8) ■ Byte 3: Track Position (bits 7:0) 	0x04
0x01	Track index	The index of the currently playing track. <ul style="list-style-type: none"> ■ Byte 0: Track Index (bits 31:24) ■ Byte 1: Track Index (bits 23:16) ■ Byte 2: Track Index (bits 15:8) ■ Byte 3: Track Index (bits 7:0) 	0x04

Event number	Event	Event data	Data length in bytes
0x02	Chapter information	<p>Chapter information, including the track index, chapter count, and chapter index.</p> <ul style="list-style-type: none"> Bytes 0–3 specify the currently playing track index, as follows: <ul style="list-style-type: none"> Byte 0: Track Index (bits 31:24) Byte 1: Track Index (bits 23:16) Byte 2: Track Index (bits 15:8) Byte 3: Track Index (bits 7:0) Bytes 4–5 specify the chapter count of the track, as follows. A value of 0x0000 indicates that the track does not have chapters. <ul style="list-style-type: none"> Byte 4: Chapter Count (bits 15:8) Byte 5: Chapter Count (bits 7:0) Bytes 6–7 specify the chapter index, as follows. A value of 0xFFFF indicates that the track does not have chapters. <ul style="list-style-type: none"> Byte 6: Chapter Index (bits 15:8) Byte 7: Chapter Index (bits 7:0) 	0x08
0x03	Play status	<p>The current play status of the Apple device; that is, whether it is playing, paused, stopped, fast forwarding or rewinding. Possible values are described in Table 3-65 (page 243).</p> <ul style="list-style-type: none"> Byte 0: Play Status (bits 7:0) 	0x01
0x04	Mute/UI Volume	<p>The current state of the mute setting and UI volume information.</p> <ul style="list-style-type: none"> Byte 0: Mute State (bits 7:0) <p>A value of 0 indicates that mute is off; a value of 1 indicates that mute is on.</p> <ul style="list-style-type: none"> Byte 1: UI Volume Level (bits 7:0) <p>A value between 0 and 255, with 0 indicating minimum volume and 255 indicating maximum volume.</p> <p>Note that if the Mute State value is true (mute is on), the UI volume level field is not valid and is returned as 0.</p>	0x02

Event number	Event	Event data	Data length in bytes
0x05	Power/battery	<p>Information about the power and battery status.</p> <ul style="list-style-type: none"> ■ Byte 0: Power State (bits 7:0) A value indicating the current power source and its state. See Table 3-68 (page 244) for possible values. ■ Byte 1: Battery Level (bits 7:0) Specifies the current battery level. A value from 0 to 255, with 0 indicating a fully discharged battery and 255 indicating a battery that is fully charged. <p>If an external power status is returned, the battery level is invalid and is returned as 0.</p>	0x02
0x06	Equalizer state	<p>The current Equalizer Setting index.</p> <ul style="list-style-type: none"> ■ Byte 0: Equalizer index (bits 31:24) ■ Byte 1: Equalizer index (bits 23:16) ■ Byte 2: Equalizer index (bits 15:8) ■ Byte 3: Equalizer index (bits 7:0) 	0x04
0x07	Shuffle	<p>The state of the shuffle setting. See Table 3-66 (page 243) for a list of possible values.</p> <ul style="list-style-type: none"> ■ Byte 0: Shuffle State (bits 7:0) 	0x01
0x08	Repeat	<p>The state of the repeat setting. See Table 3-67 (page 243) for a list of possible values.</p> <ul style="list-style-type: none"> ■ Byte 0: Repeat State (bits 7:0) 	0x01

Event number	Event	Event data	Data length in bytes
0x09	Date/time	<p>The current date and time.</p> <ul style="list-style-type: none"> ■ Bytes 0–1 specify the current year. A value of 2005 represents the year 2005 A.D. <ul style="list-style-type: none"> Byte 0: Year (bits 15:8) Byte 1: Year (bits 7:0) ■ Byte 2: Month (bits 7:0) <ul style="list-style-type: none"> A value between 1 and 12, where 1 = January and 12 = December. ■ Byte 3: Day of the month (bits 7:0) <ul style="list-style-type: none"> A value between 1 and 31. ■ Byte 4: Hour (bits 7:0) <ul style="list-style-type: none"> A value between 0 and 23, where 0 = 12:00 a.m. and 23 = 11:00 p.m. ■ Byte 5: Minute (bits 7:0) <ul style="list-style-type: none"> A value between 0 and 59. 	0x06
0x0A	Alarm	Deprecated; do not use	0x03
0x0B	Backlight	<p>The current backlight level. A value between 0 and 255, where 0 indicates the backlight is at minimum brightness and 255 indicates that the backlight is at full intensity.</p> <ul style="list-style-type: none"> ■ Byte 0: Backlight Level (bits 7:0) 	0x01
0x0C	Hold switch	<p>The current state of the hold switch. A value of 0 means the hold switch is off. A value of 1 indicates it is on.</p> <ul style="list-style-type: none"> ■ Byte 0: Hold Switch State (bits 7:0) 	0x01
0x0D	Sound check	<p>The state of the sound check setting. A value of 0 means that sound check is off; a value of 1 indicates it is on.</p> <ul style="list-style-type: none"> ■ Byte 0: Sound Check State (bits 7:0) 	0x01
0x0E	Audiobook	<p>The audiobook playback speed setting. See Table 3-69 (page 244) for a list of possible values.</p> <ul style="list-style-type: none"> ■ Byte 0: Audiobook Playback Speed Setting (bits 7:0) 	0x01
0x0F	Track position in seconds	<p>The new track time position, in seconds.</p> <ul style="list-style-type: none"> ■ Byte 0: Track Position (bits 15:8) ■ Byte 1: Track Position (bits 7:0) 	0x02

Event number	Event	Event data	Data length in bytes
0x10	Mute/UI/Absolute Volume	<p>The current state of the mute setting, UI volume, and absolute volume.</p> <ul style="list-style-type: none"> ■ Byte 0: Mute state (bits 7:0). A value of 0 indicates that muting is off; a value of 1 indicates that muting is on. ■ Byte 1: UI volume level (bits 7:0). A value between 0 and 255, normalized to volume limit settings. 0 indicates minimum volume and 255 indicates maximum volume. ■ Byte 2: Absolute volume level (bits 7:0). A value between 0 and 255, not normalized. 0 indicates minimum absolute volume and 255 indicates maximum absolute volume. If muting is on (byte 0 = 0x00), the absolute volume level is not valid and is returned as 0. 	0x03
0x11	Track capabilities	<p>Track capabilities bits:</p> <ul style="list-style-type: none"> ■ Bit 0: If equal to 1, the track is an audiobook. ■ Bit 1: If equal to 1, the track has chapters. ■ Bit 2: Set to 1 if album artwork is available, 0 otherwise. ■ Bit 3: If equal to 1, the track has song lyrics. ■ Bit 4: If equal to 1, the track is a podcast episode. ■ Bit 5: Track has release date. ■ Bit 6: Track has description. ■ Bit 7: Track contains video (a video podcast, music video, movie, or TV show). ■ Bit 8: Track is currently queued to play as a video. ■ Bit 9-12: Reserved. ■ Bit 13: Track is capable of generating a Genius playlist. ■ Bit 14: If equal to 1, the track is an iTunesU episode. ■ Bit 31:15: Reserved. 	0x04
0x12	Playback engine contents changed	Bytes 0-3: number of tracks in new playlist.	0x04
0x13–0xFF	Reserved	N/A	N/A

The battery and volume UI levels can both range from 0 (lowest) to 255 (highest). The absolute volume level ranges between 0 and the Apple device's Volume Limit setting. The UI volume is scaled to cover the Absolute volume range; thus, setting the UI volume to 255 will result in the Absolute volume being set to the Apple device's Volume Limit setting. The granularity of minimum to maximum range steps varies between Apple devices.

[Table 3-65](#) (page 243) lists the possible values for the data associated with a Play Status event and their meanings.

Table 3-65 Play status values

Value	Meaning
0x00	Playback stopped
0x01	Playing (for " Command 0x0E: SetiPodStateInfo " (page 249), start or resume playback)
0x02	Playback paused
0x03	Fast forward (FF)
0x04	Fast rewind (REW)
0x05	End fast forward or rewind mode
0x06–0xFF	Reserved

Note: With the iPhone, incoming phone calls can pause audio playback at any time. The accessory should enable notifications for this event and act accordingly (for example, by changing an icon). It should not try to cancel the pause.

[Table 3-66](#) (page 243) lists the possible values for the data associated with a Shuffle event.

Table 3-66 Shuffle state

Value	Meaning
0x00	Shuffle off
0x01	Shuffle tracks and songs
0x02	Shuffle albums
0x03–0xFF	Reserved

[Table 3-67](#) (page 243) lists the possible values for the data associated with a Repeat event.

Table 3-67 Repeat state

Value	Meaning
0x00	Repeat off
0x01	Repeat one track or song
0x02	Repeat all tracks
0x03–0xFF	Reserved

[Table 3-68](#) (page 244) lists the possible values for the data associated with a Power/Battery event and their meanings.

Table 3-68 Power and battery state

Value	Meaning
0x00	Internal battery power, low power (< 30%)
0x01	Internal battery power
0x02	External power, battery pack, no charging
0x03	External power, no charging
0x04	External power, battery charging
0x05	External power, battery charged
0x06–0xFF	Reserved

[Table 3-69](#) (page 244) lists the possible values for the data associated with an Audiobook event and their meanings.

Table 3-69 Audiobook playback speeds

Value	Meaning
0xFF	Slower (–1)
0x00	Normal
0x01	Faster (+1)
0x02–0xFE	Reserved

Command 0x0A: GetRemoteEventStatus

Direction: Accessory to Apple device

The accessory requests the status of state information that has changed on the Apple device. In response, the Apple device sends "[Command 0x0B: RetRemoteEventStatus](#)" (page 245), containing a bitmask of event states that changed since the last `GetRemoteEventStatus` command and clears all the remote event status bits. This command may be used to poll the Apple device for certain event changes without enabling asynchronous remote event notification.

Note: Accessories must not poll for play status; they must use the iAP notification mechanism instead. See ["Command 0x08: SetRemoteEventNotification"](#) (page 235).

Table 3-70 GetRemoteEventStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0A	Command ID: GetRemoteEventStatus
5	0xF1	Checksum

Command 0x0B: RetRemoteEventStatus

Direction: Apple device to Accessory

The Apple device sends this command in response to ["Command 0x0A: GetRemoteEventStatus"](#) (page 244).

Table 3-71 RetRemoteEventStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0B	Command ID: RetRemoteEventStatus
5	0xNN	remEventStatus (bits 31:24). The event status that has changed. See Table 3-62 (page 236) for a list of possible events.
6	0xNN	remEventStatus (bits 23:16)
7	0xNN	remEventStatus (bits 15:8)
8	0xNN	remEventStatus (bits 7:0)
9	0xNN	Checksum

The bits in [Table 3-62](#) (page 236) represent the events whose status has changed on the Apple device since the last `GetRemoteEventStatus` command was received. For example, if the returned `remEventStatus` field has bits 2 and 7 set, then the chapter index and shuffle states of the Apple device have changed since the last `GetRemoteEventStatus` command was sent by the accessory. Accessories can use "[Command 0x0C: GetiPodStateInfo](#)" (page 246) to get the updated state information for those events.

Command 0x0C: GetiPodStateInfo

Direction: Accessory to Apple device

The accessory obtains Apple device state information. The information type (`infoType`) field specifies the type of information to get. In response, the Apple device sends "[Command 0x0D: RetiPodStateInfo](#)" (page 247) with the requested state information.

Table 3-72 `GetiPodStateInfo` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0C	Command ID: <code>GetiPodStateInfo</code>
5	0xNN	<code>infoType</code> (1 byte). The type of state information for which to query the Apple device. See Table 3-73 (page 246).
6	0xNN	Checksum

[Table 3-73](#) (page 246) lists the different types of information for which you can query the Apple device.

Table 3-73 `infoType` values

Value	Type of information
0x00	Track time position in milliseconds
0x01	Track playback index
0x02	Chapter information
0x03	Play status (play, pause, stop, FF, and RW)
0x04	Mute and UI volume information
0x05	Power and battery status
0x06	Equalizer setting

Value	Type of information
0x07	Shuffle setting
0x08	Repeat setting
0x09	Date and time
0x0A	Alarm state and time Deprecated; do not use
0x0B	Backlight level. The returned value may differ from the value set by <code>SetiPodStateInfo</code> because of the way that an Apple device must normalize it.
0x0C	Hold switch state
0x0E	Audiobook speed
0x0F	Track time position in seconds
0x10	Mute/UI/Absolute volume
0x11	Track capabilities
0x12–0xFF	Reserved

Note: An accessory should request `infoType` values of 0x04 or 0x10 only if it can respond appropriately to the corresponding information.

For example, to retrieve the Apple device's Equalizer Setting, an accessory could send the command shown in [Table 3-74](#) (page 247).

Table 3-74 `GetiPodStateInfo` packet to retrieve the Apple device Equalizer Setting

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0C	Command ID: <code>GetiPodStateInfo</code>
5	0x06	<code>infoType</code> = 0x06 (Equalizer setting)
6	0xE8	Checksum

Command 0x0D: `RetiPodStateInfo`

Direction: Apple device to Accessory

The Apple device sends this command in response to "Command 0x0C: GetiPodStateInfo" (page 246). The format of the returned state information depends on the type of information.

Table 3-75 RetiPodStateInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0D	Command ID: RetiPodStateInfo
5	0xNN	infoType (1 byte). The type of Apple device state information returned; see Table 3-64 (page 238).
6 ... N	0xNN	infoData (variable length). The Apple device state information; see Table 3-64 (page 238).
(last byte)	0xNN	Checksum

For example, an accessory requesting the Chapter Info of the currently playing track would receive a RetiPodStateInfo command packet similar to this (assuming a track index of 10, a chapter count of 8 and a current chapter index of 3):

Table 3-76 RetiPodStateInfo packet for requesting chapter information

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0B	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0D	Command ID: RetiPodStateInfo
5	0x02	infoType = 0x02 (Chapter Information)
6	0x00	infoData = Track Index (bits 31:24)
7	0x00	infoData = Track Index (bits 23:16)
8	0x00	infoData = Track Index (bits 15:8)
9	0x0A	infoData = Track Index (bits 7:0)
10	0x00	infoData = Chapter Count (bits 15:8)

Byte number	Value	Comment
11	0x08	infoData = Chapter Count (bits 7:0)
12	0x00	infoData = Chapter Index (bits 15:8)
13	0x03	infoData = Chapter Index (bits 7:0)
14	0xCE	Checksum

Command 0x0E: SetiPodStateInfo

Direction: Accessory to Apple device

Sets the Apple device state. The information type (`infoType`) field specifies the type of information to update. In response, the Apple device sends an ACK command with the results of the operation. Some commands include a `bRestoreOnExit` parameter that optionally allows the original Apple device setting to be restored on exit.

Table 3-77 SetiPodStateInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0E	Command ID: SetiPodStateInfo
5	0xNN	infoType (1 byte). The type of Apple device state information to set.
6 ... N	0xNN	infoData (variable length). The data for the Apple device state information to set.
(last byte)	0xNN	Checksum

Table 3-78 (page 250) lists the possible values of the `infoType` field and the corresponding data in the `infoData` field.

Note: Information type 0x09 (date/time) cannot be set on an iOS device.

Table 3-78 Apple device state data

Information type		Information data	
Value	Name	Description	Length (bytes)
0x00	Track position	The new position of the track, in milliseconds. <ul style="list-style-type: none"> ■ Byte 0: Track Position (bits 31:24) ■ Byte 1: Track Position (bits 23:16) ■ Byte 2: Track Position (bits 15:8) ■ Byte 3: Track Position (bits 7:0) 	0x04
0x01	Track index	The index of the track to play. <ul style="list-style-type: none"> ■ Byte 0: Track Index (bits 31:24) ■ Byte 1: Track Index (bits 23:16) ■ Byte 2: Track Index (bits 15:8) ■ Byte 3: Track Index (bits 7:0) 	0x04
0x02	Chapter index	The new chapter index. <ul style="list-style-type: none"> ■ Byte 0: Chapter Index (bits 15:8) ■ Byte 1: Chapter Index (bits 7:0) 	0x02
0x03	Play status	The play status of the Apple device (play, pause, stop, FF or REW). See Table 3-65 (page 243) for a list of possible values. Byte 0: Play Status (bits 7:0)	0x01
0x04	Mute/UI volume	The new mute setting or UI volume level. <ul style="list-style-type: none"> ■ Byte 0: Mute State (bits 7:0) A value of 0x00 turns mute off; a value of 0x01 turns on mute. ■ Byte 1: UI volume Level (bits 7:0) A value between 0 and 255, with 0 indicating minimum UI volume and 255 indicating maximum UI volume. ■ Byte 2: bRestoreOnExit (bits 7:0) See Table 3-79 (page 253). If the mute state is 0x01, the UI volume level field is ignored.	0x03
0x05	Power/battery	Reserved: Power and battery state cannot be set.	N/A

Information type		Information data	
Value	Name	Description	Length (bytes)
0x06	Equalizer state	<p>The new Equalizer Setting.</p> <ul style="list-style-type: none"> ■ Byte 0: Equalizer index (bits 31:24) ■ Byte 1: Equalizer index (bits 23:16) ■ Byte 2: Equalizer index (bits 15:8) ■ Byte 3: Equalizer index (bits 7:0) ■ Byte 4: bRestoreOnExit (bits 7:0) <p>See Table 3-79 (page 253).</p>	0x05
0x07	Shuffle	<p>The new state of the shuffle setting.</p> <ul style="list-style-type: none"> ■ Byte 0: Shuffle State (bits 7:0) <p>See Table 3-66 (page 243) for a list of possible values.</p> <ul style="list-style-type: none"> ■ Byte 1: bRestoreOnExit (bits 7:0) <p>See Table 3-79 (page 253).</p>	0x02
0x08	Repeat	<p>The new state of the repeat setting</p> <ul style="list-style-type: none"> ■ Byte 0: Repeat State (bits 7:0) <p>See Table 3-67 (page 243) for a list of possible values.</p> <ul style="list-style-type: none"> ■ Byte 1: bRestoreOnExit (bits 7:0) <p>See Table 3-79 (page 253).</p>	0x02
0x09	Date/time	<p>The new date and time.</p> <ul style="list-style-type: none"> ■ Bytes 0–1 specify the current year. A value of 2005 represents the year 2005 A.D. <p>Byte 0: Year (bits 15:8) Byte 1: Year (bits 7:0)</p> <ul style="list-style-type: none"> ■ Byte 2: Month (bits 7:0) <p>A value between 1 and 12, where 1 = January and 12 = December.</p> <ul style="list-style-type: none"> ■ Byte 3: Day of the month (bits 7:0) <p>A value between 1 and 31.</p> <ul style="list-style-type: none"> ■ Byte 4: Hour (bits 7:0) <p>A value between 0 and 23, where 0 = 12:00 a.m. and 23 = 11:00 p.m.</p> <ul style="list-style-type: none"> ■ Byte 5: Minute (bits 7:0) <p>A value between 0 and 59.</p>	0x06

Information type		Information data	
Value	Name	Description	Length (bytes)
0x0A	Alarm	Deprecated; do not use	0x04
0x0B	Backlight	<p>The new level of the backlight.</p> <ul style="list-style-type: none"> Byte 0: Backlight Level (bits 7:0) <p>The backlight level over a range of 0 (minimum brightness) to 255 (full intensity).</p> <ul style="list-style-type: none"> Byte 1: Reserved; set to 1 (bits 7:0) 	0x02
0x0C	Hold switch	Reserved. The state of the Hold switch cannot be set.	N/A
0x0D	Sound check	<p>The new sound check state.</p> <ul style="list-style-type: none"> Byte 0: Sound Check State (bits 7:0) <p>A value of 0x00 turns sound check off; a value of 0x01 turns it on.</p> <ul style="list-style-type: none"> Byte 1: bRestoreOnExit (bits 7:0) <p>See Table 3-79 (page 253).</p>	0x02
0x0E	Audiobook speed	<p>The audiobook playback speed.</p> <ul style="list-style-type: none"> Byte 0: Audiobook Playback Speed Setting (bits 7:0) <p>See Table 3-69 (page 244) for a list of possible values.</p> <ul style="list-style-type: none"> Byte 1: bRestoreOnExit (bits 7:0) <p>See Table 3-79 (page 253).</p>	0x02
0x0F	Track position in seconds	<p>The current track time position, in seconds.</p> <ul style="list-style-type: none"> Byte 0: Track Time Position (bits 15:8) Byte 1: Track Time Position (bits 7:0) 	0x02

Information type		Information data	
Value	Name	Description	Length (bytes)
0x10	Mute/UI/Absolute volume	<p>The current state of the mute setting, UI volume, and Absolute volume.</p> <ul style="list-style-type: none"> ■ Byte 0: Mute state (bits 7:0). A value of 0 indicates that muting is off; a value of 1 indicates that muting is on. ■ Byte 1: UI volume level (bits 7:0). A value between 0 and 255, normalized to UI volume limit settings. 0 indicates minimum UI volume and 255 indicates maximum UI volume. If the accessory sets this byte to 0, the Apple device uses the Absolute volume setting. ■ Byte 2: Absolute volume level (bits 7:0). A value between 0 and 255, not normalized. 0 indicates minimum Absolute volume and 255 indicates maximum Absolute volume. If muting is on (byte 0 = 0x00), the Absolute volume level is not valid and is returned as 0. ■ Byte 3: bRestoreOnExit (bits 7:0). See Table 3-79 (page 253) for a list of values. 	0x04
0x11–0xFF	Reserved	N/A	N/A

[Table 3-79](#) (page 253) lists the possible values for the bRestoreOnExit parameter.

Table 3-79 Restore-on-exit values

Value	Meaning
0x00	Do not save the original state.
0x01	Save the original state and restore it on exit.

For example, an accessory could send the command shown in [Table 3-80](#) (page 253) to set the currently playing track on the Apple device to track 1000.

Table 3-80 SetiPodStateInfo packet for setting the current track

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0E	Command ID: SetiPodStateInfo

Byte number	Value	Comment
5	0x01	infoType = 0x01 (Track Index)
6	0x00	infoData = Track Index (bits 31:24). Sets the playback track index to 1000.
7	0x00	infoData = Track Index (bits 23:16)
8	0x03	infoData = Track Index (bits 15:8)
9	0xE8	infoData = Track Index (bits 7:0)
10	0xFC	Checksum

Note: SetiPodStateInfo commands that use information types that have data fields larger than one byte must be sure to send the data in big-endian format. See the example in [Table 3-80](#) (page 253).

Command 0x0F: GetPlayStatus

Direction: Accessory to Apple device

The accessory requests the current Apple device play status information. In response, the Apple device sends "[Command 0x10: RetPlayStatus](#)" (page 254) with the current play state, track index, track position, and track length.

Table 3-81 GetPlayStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0F	Command ID: GetPlayStatus
5	0xEC	Checksum

Command 0x10: RetPlayStatus

Direction: Apple device to Accessory

The Apple device sends this command in response to "[Command 0x0F: GetPlayStatus](#)" (page 254) and returns the current Apple device play status information. If the Apple device is in a playing or paused state, the track index (`trackIndex`), track length (`trackTotMs`), and track position (`trackPosMs`) fields are valid. Otherwise, they should be ignored.

Table 3-82 RetPlayStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0F	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x10	Command ID: RetPlayStatus
5	0xNN	playState (1 byte). The Apple device playback engine state. See Table 3-65 (page 243) for a list of possible values. If the value of this field is 0x00, all of the subsequent track fields are invalid. The value 0x05 is reserved.
6	0xNN	trackIndex (bits 31:24). Specifies the index of the currently playing track.
7	0xNN	trackIndex (bits 23:16)
8	0xNN	trackIndex (bits 15:8)
9	0xNN	trackIndex (bits 7:0)
10	0xNN	trackTotMs (bits 31:24). Specifies the total length of the track, in milliseconds.
11	0xNN	trackTotMs (bits 23:16)
12	0xNN	trackTotMs (bits 15:8)
13	0xNN	trackTotMs (bits 7:0)
14	0xNN	trackPosMs (bits 31:24). The current position of the track, in milliseconds.
15	0xNN	trackPosMs (bits 23:16)
16	0xNN	trackPosMs (bits 15:8)
17	0xNN	trackPosMs (bits 7:0)
18	0xNN	Checksum

Command 0x11: SetCurrentPlayingTrack

Direction: Accessory to Apple device

The accessory sets the Apple device's currently playing track to the track at the specified index. The total number of playing tracks can be obtained by sending "[Command 0x14: GetNumPlayingTracks](#)" (page 259). The playing track index is zero based, so the valid range is from 0x0 to numPlayTracks-1 (one less than the total count).

Table 3-83 SetCurrentPlayingTrack packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x11	Command ID: SetCurrentPlayingTrack
5	0xNN	trackIndex (bits 31:24). The track index to begin playing.
6	0xNN	trackIndex (bits 23:16)
7	0xNN	trackIndex (bits 15:8)
8	0xNN	trackIndex (bits 7:0)
9	0xNN	Checksum

Command 0x12: GetIndexedPlayingTrackInfo

Direction: Accessory to Apple device

The accessory requests track information for the specified playing track index. The `infoType` field specifies the type of information to be returned, such as track title, artist name, album name, track genre, and track chapter information. In response, the Apple device sends "Command 0x13: RetIndexedPlayingTrackInfo" (page 257) with the requested track information.

Table 3-84 GetIndexedPlayingTrackInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x09	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x12	Command ID: GetIndexedPlayingTrackInfo
5	0xNN	infoType (1 byte). The type of track information to retrieve. See Table 3-86 (page 258) for a list of the available types of information.
6	0xNN	trackIndex (bits 31:24). The index of the track for which to retrieve information.
7	0xNN	trackIndex (bits 23:16)

Byte number	Value	Comment
8	0xNN	trackIndex (bits 15:8)
9	0xNN	trackIndex (bits 7:0)
10	0xNN	chapIndex (bits 15:8). The index of the chapter for which to retrieve information. This field is valid only when infoType is chapter information (0x01) and the track at trackIndex has chapters. Set the chapIndex fields to 0x00 if infoType is not chapter information and trackIndex does not have chapters.
11	0xNN	chapIndex (bits 7:0)
12	0xNN	Checksum

Command 0x13: RetIndexedPlayingTrackInfo

Direction: Apple device to Accessory

The Apple device sends this command in response to "[Command 0x12: GetIndexedPlayingTrackInfo](#)" (page 256). It returns the requested type of information and data for the specified playing track. Data returned as strings are encoded as null-terminated UTF-8 character arrays. If the track information string does not exist, an empty string is returned. The accessory must not assume that such an empty string will be null-terminated.

Table 3-85 RetIndexedPlayingTrackInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x13	Command ID: RetIndexedPlayingTrackInfo
5	0xNN	infoType. The type of track information being returned. See Table 3-86 (page 258) for possible values.
6 ... N	0xNN	infoData The track information data. The Apple device sends a small or large command packet, depending both on the data size and the accessory's maximum packet size. Accessories must be able to handle either small or large packets containing data up to the maximum length of the packet minus packet overhead.
(last byte)	0xNN	Checksum

[Table 3-86](#) (page 258) shows the data returned for each type of track information.

Table 3-86 Track information data

Info type	Data type	Track information data	Data length
0x00	Track caps/info	<p>Track capabilities and information. This contains 3 fields:</p> <ul style="list-style-type: none"> ■ Bytes 0–3: Track Caps bitfields. Specifies the capabilities of the track. These bits have the following meanings: Bit 0: If equal to 1, the track is an audiobook. Bit 1: If equal to 1, the track has chapters. Bit 2: Set to 1 if album artwork is available, 0 otherwise. Bit 3: If equal to 1, the track has song lyrics. Bit 6:4: Reserved. Bit 7: Track contains video (a video podcast, music video, movie, or TV show). Bit 8: Track is currently queued to play as a video. Bit 12:9: Reserved. Bit 13: Track is capable of generating a Genius playlist. Bit 14: Track is an iTunesU episode. Bit 31:15: Reserved. ■ Bytes 4–7: TrackTotMs. The total length of the track in milliseconds. ■ Bytes 8–9: ChapCount. If the track has chapters, the chapter count. 	10
0x01	Chapter time/name	<p>Chapter time and name, having two fields:</p> <ul style="list-style-type: none"> ■ Bytes 0–3: Chapter Time The chapter time in milliseconds. A value of 0 indicates there are no chapters. ■ (variable length): Chapter Name The chapter name, as a UTF-8 string. 	4 + Variable
0x02	Artist name	The artist name, as a UTF-8 string.	Variable
0x03	Album name	The album name, as a UTF-8 string.	Variable
0x04	Genre name	The genre name, as a UTF-8 string.	Variable
0x05	Track title	The title of the track, as a UTF-8 string.	Variable
0x06	Composer name	The composer name, as a UTF-8 string.	Variable

Info type	Data type	Track information data	Data length
0x07	Lyrics	Track lyrics data, consisting of 3 fields: <ul style="list-style-type: none"> ■ Byte 0: Packet information bits. If set, these bits have the following meanings: <ul style="list-style-type: none"> Bit 0: If equal to 1, indicates that this is one of multiple packets. Bit 1: If equal to 1, this is the last packet (applicable only if bit 0 is equal to 1). Bits 7:2: Reserved; set to 0. ■ Bytes 1–2: Packet index (16-bit big-endian format) ■ Bytes 3–NN: Track lyrics as a UTF-8 string (see Note below). 	Variable
0x08	Artwork count	Artwork count data. The artwork count is a sequence of 4-byte records; each record consists of a 2-byte <code>formatID</code> value followed by a 2-byte count of images in that format for this track. If the track contains no format IDs, this info type will return only 1 byte containing 0x08 and no records. For information about <code>formatID</code> values, see "Command 0x17: RetArtworkFormats" (page 261).	Variable
0x09–0xFF	Reserved	N/A	N/A

Notes: iOS devices do not return track lyrics.

With other Apple devices, track lyrics are formatted as a single null-terminated UTF8 string. If the lyrics string is too long to be carried within a single packet, then the string is broken into sections and carried within separate packets with distinct values for each section index. The last lyrics packet section contains the null terminator character for the full track lyrics string. Lyric sections before the last section do not include null terminators, and individual sections may not necessarily be valid UTF8 strings. Accessories must use the packet payload length to determine the length of each string section and assemble the full lyrics string by concatenating the individual substrings in order.

If `GetIndexedPlayingTrackInfo` specified a track not currently playing, a null string is returned. Lyrics for a track being played may take up to 5 seconds to return. An accessory may try to get the current lyrics every 0.5 seconds until either 5 seconds has elapsed or the track lyrics string has been returned.

Command 0x14: GetNumPlayingTracks

Direction: Accessory to Apple device

The accessory requests the total number of tracks playing in the Apple device playback engine. The count can be used to select a different playing track or obtain information for a specific track. In response, the Apple device sends ["Command 0x15: RetNumPlayingTracks"](#) (page 260).

Table 3-87 GetNumPlayingTracks packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x14	Command = 0x14 (GetNumPlayingTracks)
5	0xE7	Checksum

Command 0x15: RetNumPlayingTracks

Direction: Apple device to Accessory

The Apple device sends this command in response to "[Command 0x14: GetNumPlayingTracks](#)" (page 259) received from the accessory. It returns the total number of tracks queued in the playback engine.

Table 3-88 RetNumPlayingTracks packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x15	Command ID: RetNumPlayingTracks
5	0xNN	numPlayTracks (bits 31:24). The total number of queued playing tracks.
6	0xNN	numPlayTracks (bits 23:16)
7	0xNN	numPlayTracks (bits 15:8)
8	0xNN	numPlayTracks (bits 7:0)
9	0xNN	Checksum

Command 0x16: GetArtworkFormats

Direction: Accessory to Apple device

The accessory sends this command to obtain the list of supported artwork formats on the Apple device. No parameters are sent. See ["Transferring Album Art"](#) (page 229).

Table 3-89 GetArtworkFormats packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x03	Lingo ID: Display Remote lingo
4	0x16	Command ID: GetArtworkFormats
5	0xE5	Checksum

Command 0x17: RetArtworkFormats

Direction: Apple device to Accessory

The Apple device sends this command to the accessory in response to ["Command 0x16: GetArtworkFormats"](#) (page 260). Each format is described in a 7-byte record (formatID:2, pixelFormat:1, width:2, height:2). The formatID is used when sending GetTrackArtworkTimes. The accessory may return zero records if the Apple device does not contain any artwork. See ["Transferring Album Art"](#) (page 229).

Table 3-90 RetArtworkFormats packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x03	Lingo ID: Display Remote lingo
4	0x17	Command ID: RetArtworkFormats
NN	0xNN	formatID (bits 15:8) Apple device-assigned value for this format
NN	0xNN	formatID (bits 7:0)
NN	0xNN	pixelFormat. See Table 3-91 (page 262).
NN	0xNN	imageWidth (bits 15:8). Number of pixels wide for each image.
NN	0xNN	imageWidth (bits 7:0)

Byte number	Value	Comment
<i>NN</i>	<i>0xNN</i>	imageHeight (bits 15:8). Number of pixels high for each image.
<i>NN</i>	<i>0xNN</i>	imageHeight (bits 7:0)
Previous 7 bytes may be repeated <i>NN</i> times		
<i>NN</i>	<i>0xNN</i>	Checksum

Table 3-91 Display pixel format codes

Display pixel format	Code
Reserved	0x00
Monochrome, 2 bits per pixel	0x01
RGB 565 color, little-endian, 16 bpp	0x02
RGB 565 color, big-endian, 16 bpp	0x03
Reserved	0x04–0xFF

Command 0x18: GetTrackArtworkData

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to request the artwork image bitmap data for a given trackIndex, formatID, and artworkIndex. See ["Transferring Album Art"](#) (page 229).

Table 3-92 GetTrackArtworkData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0C	Length of packet
3	0x03	Lingo ID: Display Remote lingo
4	0x18	Command ID: GetTrackArtworkData
5	<i>0xNN</i>	trackIndex (bits 31:24).
6	<i>0xNN</i>	trackIndex (bits 23:16)
7	<i>0xNN</i>	trackIndex (bits 15:8)
8	<i>0xNN</i>	trackIndex (bits 7:0)

Byte number	Value	Comment
9	0xNN	formatID (bits 15:8)
10	0xNN	formatID (bits 7:0)
11	0xNN	Time offset in milliseconds (bits 31:24)
12	0xNN	Time offset in milliseconds (bits 23:16)
13	0xNN	Time offset in milliseconds (bits 15:8)
14	0xNN	Time offset in milliseconds (bits 7:0)
15	0xNN	Checksum

Command 0x19: RetTrackArtworkData

Direction: Apple device to Accessory

The Apple device sends this command in response to a ["Command 0x18: GetTrackArtworkData"](#) (page 262) command received from an accessory. Multiple `RetTrackArtworkData` commands may be necessary to transfer all the data because it will be too much to fit into a single packet. See ["Transferring Album Art"](#) (page 229).

This command returns the overall image width and height in pixels, the image row size in bytes, and the inset rectangle that contains the actual artwork content. The inset rectangle coordinates consist of two *x,y* pairs. Each *x* or *y* value is 2 bytes, so the total size of the inset rectangle coordinate set is 8 bytes.

Note: Track artwork may have its own border. In this case, the inset rectangle encloses only the actual artwork inside the border, not the whole image.

The total image size in bytes is given by multiplying the row size by the image height. Padding may be inserted at the top, bottom, left, or right of the artwork to fit it to the Apple device screen.

Table 3-93 `RetTrackArtworkData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x03	Lingo ID: Display Remote lingo
4	0x19	Command ID: <code>RetTrackArtworkData</code>
5	0xNN	Descriptor packet index (15:8). These fields uniquely identify each packet in the <code>RetTrackArtworkData</code> transaction. The first packet is the descriptor packet, which always starts with an index of 0x0000.

Byte number	Value	Comment
6	0xNN	Descriptor packet index (7:0)
7	0xNN	Display pixel format code. See Table 3-91 (page 262).
8	0xNN	Image width in pixels (15:8)
9	0xNN	Image width in pixels (7:0)
10	0xNN	Image height in pixels (15:8)
11	0xNN	Image height in pixels (7:0)
12	0xNN	Inset rectangle, top-left point, x value (15:8)
13	0xNN	Inset rectangle, top-left point, x value (7:0)
14	0xNN	Inset rectangle, top-left point, y value (15:8)
15	0xNN	Inset rectangle, top-left point, y value (7:0)
16	0xNN	Inset rectangle, bottom-right point, x value (15:8)
17	0xNN	Inset rectangle, bottom-right point, x value (7:0)
18	0xNN	Inset rectangle, bottom-right point, y value (15:8)
19	0xNN	Inset rectangle, bottom-right point, y value (7:0)
20	0xNN	Row size in bytes (bits 31:24)
21	0xNN	Row size in bytes (bits 23:16)
22	0xNN	Row size in bytes (bits 15:8)
23	0xNN	Row size in bytes (bits 7:0)
24	0xNN...	Image pixel data (variable length)
NN	0xNN	Checksum

Note: In subsequent packets in the sequence (packets with a descriptor packet index greater than 0x0000), bytes 7 through 23 are omitted.

Command 0x1A: GetPowerBatteryState

Direction: Accessory to Apple device

The accessory sends this command to obtain the power and battery level state of the Apple device. In response, the Apple device sends "[Command 0x1B: RetPowerBatteryState](#)" (page 265) with the power and battery information.

Table 3-94 GetPowerBatteryState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x1A	Command ID: GetPowerBatteryState
5	0xE1	Checksum

Command 0x1B: RetPowerBatteryState

Direction: Apple device to Accessory

The Apple device sends this command in response to "[Command 0x1A: GetPowerBatteryState](#)" (page 264), returning the current Apple device power state and battery level.

Note: If an external power status (indicated by the value of the `powerStat` field) is returned, the battery level is invalid and is 0 on return.

Table 3-95 RetPowerBatteryState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x1B	Command ID: RetPowerBatteryState
5	0xNN	<code>powerStat</code> (1 byte). The battery power state. See Table 3-68 (page 244).
6	0xNN	<code>battLevel</code> (1 byte). The battery power level. This is a value between 00 (the battery is fully discharged, no power remains) and 255 (the battery is fully charged). This field is valid only if <code>powerStat</code> is Internal battery (0x00 or 0x01)
7	0xNN	Checksum

Command 0x1C: GetSoundCheckState

Direction: Accessory to Apple device

The accessory requests the Apple device's current sound check setting. When enabled, sound check adjusts track playback volume to the same level. In response, the Apple device sends "[Command 0x1D: RetSoundCheckState](#)" (page 266) with the current sound check state.

Table 3-96 GetSoundCheckState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x1C	Command ID: GetSoundCheckState
5	0xDF	Checksum

Command 0x1D: RetSoundCheckState

Direction: Apple device to Accessory

The Apple device sends this command in response to "[Command 0x1C: GetSoundCheckState](#)" (page 266) and returns the current state of the sound check setting.

Table 3-97 RetSoundCheckState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x1D	Command ID: RetSoundCheckState
5	0xNN	sndChkState (1 byte). The current state of the sound check setting. A value of 0x00 indicates that sound check is off; 0x01 indicates that it is on.
6	0xNN	Checksum

Command 0x1E: SetSoundCheckState

Direction: Accessory to Apple device

The accessory sets the state of the Apple device's sound check setting and optionally saves the previous sound check state to be restored on accessory detach. In response to this command, the Apple device sends an ACK packet with the status of the command.

Table 3-98 SetSoundCheckState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x1E	Command ID: SetSoundCheckState
5	0xNN	sndChkState (1 byte). The new state of the sound check setting. A value of 0x00 turns off sound check off; 0x01 turns on sound check.
6	0xNN	bRestoreOnExit (1 byte). Specifies whether the original sound check state is saved and restored on exit. See Table 3-79 (page 253)
7	0xNN	Checksum

Command 0x1F: GetTrackArtworkTimes

Direction: Accessory to Apple device

The accessory sends this command to the Apple device to request the list of artwork time offsets for a track. A 4-byte `trackIndex` specifies which track is to be selected. A 2-byte `formatID` indicates which type of artwork is desired. The format IDs that the Apple device supports can be obtained using the "[Command 0x16: GetArtworkFormats](#)" (page 260) command.

The 2-byte `artworkIndex` specifies where to begin searching for artwork. A value of 0 indicates that the Apple device should start with the first available artwork.

The 2-byte `artworkCount` specifies the maximum number of times to be returned. A value of -1 (0xFFFF) indicates that all artwork times from the specified index should be returned.

Table 3-99 GetTrackArtworkTimes packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x0C	Length of packet
3	0x03	Lingo ID: Display Remote lingo
4	0x1F	Command ID: GetTrackArtworkTimes
5	0xNN	trackIndex (bits 31:24)
6	0xNN	trackIndex (bits 23:16)
7	0xNN	trackIndex (bits 15:8)
8	0xNN	trackIndex (bits 7:0)
9	0xNN	formatID (bits 15:8)
10	0xNN	formatID (bits 7:0)
11	0xNN	artworkIndex (bits 15:8)
12	0xNN	artworkIndex (bits 7:0)
13	0xNN	artworkCount (bits 15:8)
14	0xNN	artworkCount (bits 7:0)
15	0xNN	Checksum

Command 0x20: RetTrackArtworkTimes

Direction: Apple device to Accessory

The Apple device sends this command to the accessory in response to a ["Command 0x1F: GetTrackArtworkTimes"](#) (page 267) command. The accessory returns zero or more 4-byte records, one for each piece of artwork associated with the track and format specified by `GetTrackArtworkTimes`. Artwork times are expressed as offsets, in milliseconds, from the beginning of the track.

The number of records returned will be no greater than the number specified in the `GetTrackArtworkTimes` command. It may, however, be less than requested. This can happen if there are fewer pieces of artwork available than were requested, or if the Apple device is unable to place the full number in a single packet. Check the number of records returned against the results of `RetIndexedPlayingTrackInfo` with `infoType 0x08` to ensure that all artwork has been received.

Table 3-100 RetTrackArtworkTimes packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0xNN	Length of packet
3	0x03	Lingo ID: Display Remote lingo
4	0x20	Command ID: RetTrackArtworkTimes
5	0xNN	Time offset in milliseconds (bits 31:24)
6	0xNN	Time offset in milliseconds (bits 23:16)
7	0xNN	Time offset in milliseconds (bits 15:8)
8	0xNN	Time offset in milliseconds (bits 7:0)
The preceding 4 bytes may be repeated NN times.		
NN	0xNN	Checksum

Command 0x21: CreateGeniusPlaylist

Direction: Accessory to Apple device

The accessory sends this command to an Apple device to ask it to create a Genius playlist. The Apple device returns a Display Remote lingo ACK command, passing one of the responses listed in [Table 3-102](#) (page 270). If the playlist is successfully created, it starts playing.

Note: This command may take up to 30 seconds to finish.

Table 3-101 CreateGeniusPlaylist command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x08	Packet payload length
3	0x03	Lingo ID: Display Remote lingo
4	0x21	Command ID: CreateGeniusPlaylist
5	0xNN	transID [bits 15:8]; see "Transaction IDs" (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	Track index (bits 31:24); Playback index
8	0xNN	Track index (bits 23:16)

Byte number	Value	Meaning
9	0xNN	Track index (bits 15:8)
10	0xNN	Track index (bits 7:0)
11	0xNN	Packet payload checksum byte

Table 3-102 ACK responses to CreateGeniusPlaylist

ACK response	Meaning
0x00	Genius playlist is being created. When the playlist is complete, the Apple device sends an <code>iPodNotification</code> command for Command Complete; see Table 2-132 (page 164).
0x02	Genius playlist could not be created.
0x04	Track index not valid.
0x12	Genius information not available for that track (see " Command 0x22: IsGeniusAvailableForTrack " (page 270)).

Command 0x22: IsGeniusAvailableForTrack

Direction: Accessory to Apple device

The accessory sends this command to an Apple device to determine if Genius information is available for a given track. The Apple device returns a Display Remote lingo ACK command, passing one of the responses listed in [Table 3-104](#) (page 271).

Table 3-103 IsGeniusAvailableForTrack command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x08	Packet payload length
3	0x03	Lingo ID: Display Remote lingo
4	0x22	Command ID: IsGeniusAvailableForTrack
5	0xNN	transID [bits 15:8]; see " Transaction IDs " (page 525).
6	0xNN	transID [bits 7:0]
7	0xNN	Track index (bits 31:24): Playback index
8	0xNN	Track index (bits 23:16)

Byte number	Value	Meaning
9	0xNN	Track index (bits 15:8)
10	0xNN	Track index (bits 7:0)
11	0xNN	Packet payload checksum byte

Table 3-104 ACK responses to `IsGeniusAvailableForTrack`

ACK response	Meaning
0x00	Genius information is available. This does not guarantee that the Apple device can create a Genius playlist, because doing so depends on factors such as the availability of matching songs in the database.
0x04	Track index not valid.
0x12	Track index is valid, but Genius information is not available for that track.

Lingo 0x04: Extended Interface Lingo

Lingo 0x04 of the iAP is specified in detail in ["The Extended Interface Protocol"](#) (page 409).

Lingo 0x05: Accessory Power Lingo

The Accessory Power lingo is used by devices that draw up to 100 mA peak current from the Apple device through pin 13 of the 30-pin connector (Accessory Power). To use this lingo, accessories must identify themselves as intermittent high-power devices during the identification process. They may include accessories that transmit an Apple device's analog audio over radio frequencies, typically over an unused frequency in the FM band.

Note: Any accessory drawing power from an Apple device must conform to the requirements of the section "Accessory Power Policy" in *MFi Accessory Hardware Specification*.

The Accessory Power lingo is intended for use in conjunction with audio playback from the Apple device. The accessory must remain in low power mode, as defined in the section "Accessory Power Policy" in *MFi Accessory Hardware Specification*, until it receives a `BeginHighPower` command, which notifies it that it may begin drawing high power. The `EndHighPower` command notifies the accessory that it must stop drawing high power and return to low power mode within 1 second of receiving the command.

Like all accessories, an accessory using the Accessory Power lingo must comply with the power requirements of the Apple device's Sleep and Hibernation states, as specified in Appendix A, "Apple Device Power States and Accessory Power," in *MFi Accessory Hardware Specification*. Accessories are informed of Apple device state changes by receiving a General lingo `NotifyiPodStateChange` command.

Table 3-105 Accessory Power lingo command summary

Command	ID	Data length	Protocol version	Authentication required
Reserved	0x00–0x01	N/A	N/A	N/A
BeginHighPower	0x02	0x00	All	UART: No USB: Yes
EndHighPower	0x03	0x00	All	
Reserved	0x04–0xFF	N/A	N/A	N/A

Command History of the Accessory Power lingo

Table 3-106 (page 272) shows the history of command changes in the Accessory Power lingo:

Table 3-106 Accessory Power lingo command history

Lingo version	Command changes	Features
(0.00)	Add: 0x02–0x03	Begin/EndHighPower support
1.00	None	Version number available through RequestLingoProtocol - Version
1.01	None	BugFix: BeginTransmission command sent after accessory inserted while the Apple device is playing

Command 0x02: BeginHighPower

Direction: Apple device to Accessory

The Apple device sends this command to notify the accessory that high power may be used. Upon receipt of the this command, the accessory may begin drawing more than low power, up to a maximum of 100 mA, if it has previously requested intermittent high power during its identification process.

Table 3-107 BeginHighPower packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x05	Lingo ID: Accessory Power lingo
4	0x02	Command ID: BeginHighPower
5	0xF7	Checksum

Command 0x03: EndHighPower

Direction: Apple device to Accessory

The Apple device sends this command to notify the accessory to stop using accessory high power. The accessory must reduce its power usage to low power within 1 second of receiving an `EndHighPower` packet.

Note: Failure to reduce current drawn from the Apple device to low power status within 1 second after receiving this notification could damage the Apple device.

Table 3-108 `EndHighPower` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x05	Lingo ID: Accessory Power lingo
4	0x03	Command ID: <code>EndHighPower</code>
5	0xF6	Checksum

Lingo 0x06: USB Host Mode Lingo

All Apple devices can operate as devices controlled by an external USB host. Some Apple devices (see [Table 1-6](#) (page 45)) can also act as hosts controlling external USB devices. This USB Host Mode is described in ["Using an Apple Device as a USB Host"](#) (page 54). By default, the Apple device boots up in USB Device mode, ready to connect to a Macintosh or Windows computer for media data transfers. If the Apple device supports this lingo, the iAP commands specified in this section may be used to change it to USB Host mode. They can be sent only by UART transport.

[Table 3-109](#) (page 273) lists the iAP commands used to manage USB Host mode.

Table 3-109 USB Host mode commands

Cmd ID	Cmd name	Direction	Parameters
0x00	<code>DevACK</code>	Acc to Dev	{cmdStatus:1, cmdIDOrig:1}
0x04	<code>NotifyUSBMode</code>	Dev to Acc	{usbMode:1}
0x80	<code>iPodACK</code>	Dev to Acc	{cmdStatus:1, cmdIDOrig:1}
0x81	<code>GetiPodUSBMode</code>	Acc to Dev	none
0x82	<code>RetiPodUSBMode</code>	Dev to Acc	{usbMode:1}

Cmd ID	Cmd name	Direction	Parameters
0x83	SetiPodUSBMode	Acc to Dev	{usbMode:1}

Command History of the USB Host Lingoes

[Table 3-110](#) (page 274) shows the history of command changes in the USB Host Control and USB Host Mode lingoes:

Table 3-110 Command history of the USB Host Control and Host Mode lingoes

Lingo name	Version	Command changes	Features
USB Host Control	1.00	Add: 0x00–0x03	USB power state support
USB Host Mode	1.00	Deprecate: 0x00–0x03	See " Deprecated USB Host Control Commands " (page 613)
		Add: new 0x00, 0x04, and 0x80–0x83	Support USB Host mode on the Apple device

Command 0x00: DevACK

Direction: Accessory to Apple device

The accessory sends this command in response to a command received from the Apple device. The accessory must send a response when a bad parameter is received, an unsupported or invalid command is received, or a command that does not return any data has finished.

Table 3-111 DevACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x06	Lingo ID: USB Host Mode lingo
4	0x00	Command ID: DevACK
5	0xNN	cmdStatus: error codes; see Table 3-112 (page 275)
6	0xNN	cmdOrig: ID of the command being acknowledged
7	0xNN	Checksum

Table 3-112 DevACK command error codes

Value	Description
0x00	Success (OK)
0x02	Command valid but did not succeed
0x03	Out of resources
0x04	Bad parameter
0x07	Not authenticated to use this lingo or command
0x08	Bad authentication version
0x09	Power mode error
0x0A-0x0E	Reserved
0x0F	Operation timed out
0x10-0xFF	Reserved

Command 0x04: NotifyUSBMode

Direction: Apple device to Accessory

The Apple device sends this command to the accessory to notify it that the Apple device's USB host/device role has changed. In response, the accessory must return a DevACK command.

Table 3-113 NotifyUSBMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x06	Lingo ID: USB Host Mode lingo
4	0x04	Command ID: NotifyUSBMode
5	0xNN	usbMode: USB mode status; see Table 3-114 (page 276)
6	0xNN	Checksum

Table 3-114 USB Host mode status codes

Value	Description
0x00	Apple device USB interface is disabled or unavailable
0x01	Apple device USB interface is in Device Mode
0x02	Apple device USB interface is in Host Mode
0x03–0xFF	Reserved

Command 0x80: iPodACK

Direction: Apple device to Accessory

The Apple device sends this command in response to a command received from the accessory. This command is sent when a bad parameter is received, an unsupported or invalid command is received, or a command that does not return any data has finished.

Table 3-115 iPodACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x06	Lingo ID: USB Host Mode lingo
4	0x80	Command ID: iPodACK
5	0xNN	cmdStatus: error codes; see Table 3-116 (page 276)
6	0xNN	cmdOrig: ID of the command being acknowledged
7	0xNN	Checksum

Table 3-116 iPodACK command error codes

Value	Description
0x00	Success (OK)
0x02	Command valid but did not succeed
0x04	Bad parameter
0x07	Not authenticated to use this lingo or command
0x09	Accessory power mode request failed

Value	Description
0x0F	Operation timed out

Command 0x81: GetiPodUSBMode

Direction: Accessory to Apple device

The accessory sends this command to determine the current Apple device USB Host mode status. In response, the Apple device returns a RetiPodUSBMode command.

Table 3-117 GetiPodUSBMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x06	Lingo ID: USB Host Mode lingo
4	0x81	Command ID: GetiPodUSBMode
5	0x77	Checksum

Command 0x82: RetiPodUSBMode

Direction: Apple device to Accessory

The Apple device sends this command in response to a GetiPodUSBMode command, passing the status of its USB Host mode.

Table 3-118 RetiPodUSBMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x06	Lingo ID: USB Host Mode lingo
4	0x82	Command ID: RetiPodUSBMode
5	0xNN	usbMode: USB mode status; see Table 3-114 (page 276)
6	0xNN	Checksum

Command 0x83: SetiPodUSBMode

Direction: Accessory to Apple device

The accessory sends this command to set a new USB Host mode on the Apple device. In response, the Apple device returns an `iPodACK` command with the status of the operation. If the Apple device is already in the requested USB mode, the `iPodACK` command returns a success (0x00) status. If the requested USB mode cannot be set (for example, if the accessory requests USB Device mode while it uses hardware to force the Apple device into USB Host mode), the `iPodACK` command returns a Command Unavailable (0x10) status.

Table 3-119 SetiPodUSBMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x06	Lingo ID: USB Host Mode lingo
4	0x83	Command ID: SetiPodUSBMode
5	0xNN	usbMode: USB mode to set; see Table 3-120 (page 278)
6	0xNN	Checksum

Table 3-120 SetiPodUSBMode codes

Value	Description
0x00	Reserved
0x01	Set Apple device USB interface to Device Mode
0x02	Set Apple device USB interface to Host Mode
0x03–0xFF	Reserved

Lingo 0x07: RF Tuner Lingo

An external radio frequency tuner accessory can be attached to an Apple device to provide radio reception. When an Apple device successfully acknowledges an accessory's declaration of the RF Tuner lingo during the identification process, it makes a radio menu item available. Choosing this menu item displays the Apple device's tuner application. The application lets the user change the Apple device's music source and control the accessory's RF band and tuner frequency.

Note: If an Apple device contains an RF Tuner (such as the FM tuner in the 5G nano), the external tuner accessory will override it.

The RF Tuner lingo is used to pass control and state information between an Apple device and an RF tuner accessory. Normally the Apple device is the master and the accessory responds to commands from the Apple device. This means that the Apple device can initiate actions such as controlling tuner power, setting the tuner's band and frequency, initiating up or down frequency scans, and so on. An Apple device attached to an RF tuner accessory also stores station frequencies and other tuner state information.

Every RF tuner accessory must report all of its capabilities back to the attached Apple device, including support for at least one of the RF bands listed in [Table 3-128](#) (page 285). Based on the capabilities reported by the RF tuner accessory, the Apple device's tuner application may choose to change its appearance to reflect the presence or absence of certain RF tuner features.

RF Tuner Accessory Design

RF tuner accessory devices for Apple devices must meet the following requirements:

- All RF tuner lingo commands require authentication.
- An Apple device can support only one RF tuner accessory at any time, and that accessory must be attached using the 30-pin connector.
- On reset or power up, the RF tuner accessory must be in low power mode, as defined in the section "Accessory Power Policy" in *MFi Accessory Hardware Specification*, with the tuner off and any audio output disabled.
- US devices must support the Europe/US FM band (87.5–108.0 MHz), with 200 KHz channel spacing and default 75 μ sec deemphasis.
- EU/US devices must support the Europe/US FM band (87.5–108.0 MHz), with 100 KHz channel spacing and selectable 50 μ sec or 75 μ sec deemphasis.
- JP devices must support the Japan FM band (76.0–90.0 MHz), with 100 KHz channel spacing and selectable 50 μ sec or 75 μ sec deemphasis.

An RF tuner accessory may support US, EU, and/or JP requirements.

The following options apply to RF tuner devices:

- An RF tuner accessory may send asynchronous notification of state changes to an attached Apple device, but such notifications are not required.
- RF tuner devices are not expected to retain state information across interruptions of current from the Apple device resulting from the invocation of Hibernate mode.
- RF tuner devices may support HD radio.

RF Tuner Power

An attached RF tuner accessory will be notified of Apple device state changes such as transitions between Power On, Sleep, and Hibernate states. It is responsible for keeping its power consumption below the maximum allowed limits for each Apple device state. For details, see “Accessory Power Policy” in *MFi Accessory Hardware Specification*.

Current from the Apple device to the accessory is completely shut off during hibernation. When an Apple device wakes from hibernation and the accessory detects current from the Apple device on the Accessory Power line of the 30-pin connector, the accessory must wait at least 80 ms and then start the IDPS process. When an Apple device transitions from sleep to power on, it sends a `RequestIdentify` command to the accessory. In either case, the accessory must reidentify and reauthenticate itself, using IDPS (or `IdentifyDeviceLingoes` if the Apple device does not support IDPS).

RF Tuner accessories can register for intermittent high power during the identification process. If properly registered, they may draw up to 100 mA after receiving a `SetTunerCtrl` command specifying power-on. They must reduce their power consumption to low power mode within 1 second of receiving a `SetTunerCtrl` command specifying power-off. Regardless of the power state specified by the Apple device, all devices must comply with the Apple device state changes cited above.

RF Tuner Lingo Commands

[Table 3-121](#) (page 280) summarizes the RF Tuner commands (lingo 0x07).

IMPORTANT: The accessory must not exceed the command timings listed in [Table 3-2](#) (page 184) when responding to RF Tuner commands from the Apple device.

Table 3-121 RF Tuner lingo command summary

Command	ID	Direction	Data length	Protocol version	Authentication required
ACK	0x00	Acc to Dev	4 or 8	1.00	Yes
GetTunerCaps	0x01	Dev to Acc	2	1.00	Yes
RetTunerCaps	0x02	Acc to Dev	8	1.00	Yes
GetTunerCtrl	0x03	Dev to Acc	2	1.00	Yes
RetTunerCtrl	0x04	Acc to Dev	3	1.00	Yes
SetTunerCtrl	0x05	Dev to Acc	3	1.00	Yes
GetTunerBand	0x06	Dev to Acc	2	1.00	Yes
RetTunerBand	0x07	Acc to Dev	3	1.00	Yes
SetTunerBand	0x08	Dev to Acc	3	1.00	Yes
GetTunerFreq	0x09	Dev to Acc	2	1.00	Yes

Command	ID	Direction	Data length	Protocol version	Authentication required
RetTunerFreq	0x0A	Acc to Dev	7	1.00	Yes
SetTunerFreq	0x0B	Dev to Acc	6	1.00	Yes
GetTunerMode	0x0C	Dev to Acc	2	1.00	Yes
RetTunerMode	0x0D	Acc to Dev	3	1.00	Yes
SetTunerMode	0x0E	Dev to Acc	3	1.00	Yes
GetTunerSeekRssi	0x0F	Dev to Acc	2	1.00	Yes
RetTunerSeekRssi	0x10	Acc to Dev	3	1.00	Yes
SetTunerSeekRssi	0x11	Dev to Acc	3	1.00	Yes
TunerSeekStart	0x12	Dev to Acc	3	1.00	Yes
TunerSeekDone	0x13	Acc to Dev	7	1.00	Yes
GetTunerStatus	0x14	Dev to Acc	2	1.00	Yes
RetTunerStatus	0x15	Acc to Dev	3	1.00	Yes
GetStatusNotifyMask	0x16	Dev to Acc	2	1.00	Yes
RetStatusNotifyMask	0x17	Acc to Dev	3	1.00	Yes
SetStatusNotifyMask	0x18	Dev to Acc	3	1.00	Yes
StatusChangeNotify	0x19	Acc to Dev	3	1.00	Yes
GetRdsReadyStatus	0x1A	Dev to Acc	2	1.00	Yes
RetRdsReadyStatus	0x1B	Acc to Dev	6	1.00	Yes
GetRdsData	0x1C	Dev to Acc	3	1.00	Yes
RetRdsData	0x1D	Acc to Dev	0xNN	1.00	Yes
GetRdsNotifyMask	0x1E	Dev to Acc	2	1.00	Yes
RetRdsNotifyMask	0x1F	Acc to Dev	6	1.00	Yes
SetRdsNotifyMask	0x20	Dev to Acc	6	1.00	Yes
RdsReadyNotify	0x21	Acc to Dev	0xNN	1.00	Yes
Reserved	0x22–0x24	N/A	N/A	N/A	N/A
GetHDPProgramServiceCount	0x25	Dev to Acc	0	1.01	Yes
RetHDPProgramServiceCount	0x26	Acc to Dev	1	1.01	Yes

Command	ID	Direction	Data length	Protocol version	Authentication required
GetHDProgramService	0x27	Dev to Acc	0	1.01	Yes
RethHDProgramService	0x28	Acc to Dev	1	1.01	Yes
SetHDProgramService	0x29	Dev to Acc	1	1.01	Yes
GetHDDDataReadyStatus	0x2A	Dev to Acc	0	1.01	Yes
RethHDDDataReadyStatus	0x2B	Acc to Dev	4	1.01	Yes
GetHDDData	0x2C	Dev to Acc	1	1.01	Yes
RethHDDData	0x2D	Acc to Dev	0xNN	1.01	Yes
GetHDDDataNotifyMask	0x2E	Dev to Acc	0	1.01	Yes
RethHDDDataNotifyMask	0x2F	Acc to Dev	4	1.01	Yes
SetHDDDataNotifyMask	0x30	Dev to Acc	4	1.01	Yes
HDDDataReadyNotify	0x31	Acc to Dev	0xNN	1.01	Yes
Reserved	0x32–0xFF	N/A	N/A	N/A	N/A

Command History of the RF Tuner Lingo

Table 3-122 (page 282) shows the history of changes to the RF Tuner lingo.

Table 3-122 RF Tuner lingo command history

Lingo version	Command changes	Features
1.00	Add: 0x00–0x21	RF tuner control and support
1.01	Update: 0x02, 0x04, 0x05, 0x07, 0x0D, 0x0E, 0x12, 0x13, 0x15, 0x17, 0x18, 0x19, 0x1B, 0x1C, 0x1D, 0x1F, 0x20, and 0x21 Add: 0x25–0x31	Support for HD radio, AM tuning, FM wide band

Command 0x00: ACK

Direction: Accessory to Apple device

This command is sent by an RF tuner accessory on completion of a command received from an Apple device. An ACK response is also sent if a bad parameter is received, an unsupported or invalid command is received, or a command that does not return any data has finished.

Note: Certain bytes of the RF tuner lingo ACK command packet are included only if the value of `cmdStatus` is 0x06. See [Table 3-123](#) (page 283) and [Table 3-124](#) (page 283).

Table 3-123 RF tuner lingo ACK packet with `cmdStatus` not 0x06

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x00	Command ID: ACK
5	0xNN	<code>cmdStatus</code> : Status of command received (see Table 3-125 (page 284))
6	0xNN	<code>cmdIDOrig</code> : ID of the command for which the response is being sent
7	0xNN	Checksum

Table 3-124 RF tuner lingo ACK packet with `cmdStatus` equal to 0x06

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x00	Command ID: ACK
5	0x06	<code>cmdStatus</code> : Command pending
6	0xNN	<code>cmdIDOrig</code> : ID of the command for which the response is being sent
7	0xNN	<code>cmdPendTime</code> (bits 31:24). Total timeout in milliseconds for the pending command to complete.
8	0xNN	<code>cmdPendTime</code> (bits 23:16)
9	0xNN	<code>cmdPendTime</code> (bits 15:8)
10	0xNN	<code>cmdPendTime</code> (bits 7:0)
11	0xNN	Checksum

Table 3-125 RF tuner lingo ACK command status values

Value	Meaning
0x00	Command OK
0x01	Unknown track category (not applicable)
0x02	Command failed (command valid but did not succeed)
0x03	Out of resources (Apple device's internal allocation failed)
0x04	Bad parameter (command or input parameters invalid)
0x05	Unknown track ID (not applicable)
0x06	Command pending (<code>cmdPendTime</code> parameter returned)
0x07	Not authenticated (Apple device not authenticated)

Note: A value of 0x06 is usually returned in the `cmdStatus` byte of the RF tuner ACK packet for commands that require more than 100 ms to complete. In this case, the `cmdPendTime` parameter is included in bytes 7–10 of the ACK packet. If the completion status is not returned by the total number of milliseconds specified by these bytes, the Apple device will assume that a command failure has occurred and will retry the command or otherwise recover from the error. The RF tuner accessory should not return any response to the command after this timeout period has expired.

Command 0x01: GetTunerCaps

Direction: Apple device to Accessory

This command is sent by an Apple device to query an attached RF tuner accessory's capabilities and determine what features the accessory supports. In response, the accessory must send a `RetTunerCaps` command with a payload specifying its capabilities.

Table 3-126 GetTunerCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x01	Command ID: GetTunerCaps

Byte number	Value	Comment
5	0xF6	Checksum

Command 0x02: RetTunerCaps

Direction: Accessory to Apple device

This command is sent by an RF tuner accessory in response to a `GetTunerCaps` command sent by an Apple device. The command transmits a payload indicating which RF tuner capabilities it supports.

Table 3-127 RetTunerCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x02	Command ID: RetTunerCaps
5	0xNN	Tuner capabilities (bits 31:24) (See Table 3-128 (page 285))
6	0xNN	Tuner capabilities (bits 23:16)
7	0xNN	Tuner capabilities (bits 15:8)
8	0xNN	Tuner capabilities (bits 7:0)
9	0x01	Reserved; set to 0x01
10	0x00	Reserved; set to 0x00
11	0xNN	Checksum

Note: The capabilities bits returned in the `RetTunerCaps` payload (bytes 5-8) correspond to the band, control, mode, and status commands of the RF tuner lingo. The Apple device will enable features or use commands only if the RF tuner accessory sets the associated capabilities bits when sending this command.

Table 3-128 RF tuner accessory capabilities payload

Bits	Capability
00	AM band, Worldwide (520-1710 KHz) capable
01	FM band, Europe/US (87.5–108.0 MHz) capable

Bits	Capability
02	FM band, Japan (76.0–90.0 MHz) capable
03	FM band, Wide (76.0–108.0 MHz) capable
04	HD Radio capable; may be set only if bit 01 is set
07:05	Reserved; set to 0
08	Tuner power on/off control capable
09	Status change notification capable
15:10	Reserved; set to 0
17:16	Minimum FM resolution ID bits (see Table 3-129 (page 286))
18	Tuner seek up/down capable
19	Tuner seek RSSI threshold capable (must not be set if bit 18 is 0)
20	Force monophonic mode capable
21	Stereo blend capable
22	FM Tuner deemphasis select capable
23	AM tuner resolution 9KHz (0=10KHz only) capable
24	Radio Data System (RDS/RBDS) data capable
25	Tuner channel RSSI indication capable
26	Stereo source indicator capable
27	RDS/RBDS Raw mode capable
31:28	Reserved; set to 0

Note: Bit 04 (HD Radio capable), listed in [Table 3-128](#) (page 285), may be set in addition to the FM band bits (00-03), but only if bit 01 is set. This bit indicates that the radio is capable of receiving HD signals on an existing Europe/US AM or FM band.

Table 3-129 Minimum FM resolution ID bits

ID bits	Description
00	200 kHz capable
01	100 kHz capable
10	50 kHz capable

ID bits	Description
11	Reserved

Note: The minimum FM resolution bits (bits 17-16) should report the smallest FM tuner resolution that the RF tuner accessory supports.

Command 0x03: GetTunerCtrl

Direction: Apple device to Accessory

An Apple device sends this command to an RF tuner accessory to get the accessory's control state. In response, the accessory must send a `RetTunerCtrl` command with its current control state.

Table 3-130 `GetTunerCtrl` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x03	Command ID: <code>GetTunerCtrl</code>
5	0xF4	Checksum

Command 0x04: RetTunerCtrl

Direction: Accessory to Apple device

An RF tuner accessory uses this command to send its current accessory control state in response to a `GetTunerCtrl` command from an Apple device. Tuner control bits may be set only if the corresponding capabilities bits have been set in a previous `RetTunerCaps` command.

Table 3-131 `RetTunerCtrl` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo

Byte number	Value	Comment
4	0x04	Command ID: RetTunerCtrl
5	0xNN	Tuner control state (see Table 3-132 (page 288))
6	0xNN	Checksum

Table 3-132 Tuner control state bits

Bit	Description
0	RF tuner accessory power draw is on (1) or off (0). When RF tuner accessory power draw is off, the accessory should rest in the lowest power state that still allows iAP commands to be received and processed.
1	Status change notification is enabled (1) or disabled (0). When status change notification is disabled, the Apple device assumes that the accessory will send no asynchronous <code>StatusChangeNotify</code> commands. The Apple device may poll the accessory for changes.
2	Reserved
3	RDS/RBDS Raw mode enabled
7:4	Reserved

Command 0x05: SetTunerCtrl

Direction: Apple device to Accessory

This command is sent by an Apple device to control an RF tuner accessory's state. In response, the accessory must return an `ACK` command with the command status. RF tuner state information, such as tuner frequency, band, and so on, must be preserved by the accessory across tuner on and off cycles, assuming current is available from the Apple device.

When the tuner power is turned off, it must disable its audio output and reduce its power consumption to low power mode, as defined in the section "Accessory Power Policy" in *MFi Accessory Hardware Specification*. When tuner power is switched on, its power consumption may rise to the maximum declared during its identification process. If an accessory has specified that it requires intermittent accessory high power (up to 100 mA while on), the Apple device will enable its accessory high power before sending a `SetTunerCtrl` command to turn on tuner power. The Apple device accessory high power will remain on until a subsequent `SetTunerCtrl` command to turn off tuner power has finished. The accessory must reduce its power consumption to low power mode within 1 second of receiving the command.

Note: The Apple device `NotifyiPodStateChange` command overrides this command for Apple device transitions to the Sleep or Hibernate state. Even if this command has let the RF tuner draw current from the Apple device, the accessory must reduce its power consumption to low power mode on receiving an Apple device state change notification that specifies a transition to Sleep or Hibernate. The only Apple device power state in which an RF tuner accessory requesting high power can consume more than low power is the Power On state.

Table 3-133 SetTunerCtrl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x05	Command ID: SetTunerCtrl
5	0xNN	Tuner control bits (see Table 3-134 (page 289))
6	0xNN	Checksum

Table 3-134 Tuner control bits

Bit	Description
0	Turn RF tuner current draw from the Apple device on (1) or off (0). When RF tuner current draw is turned off, the accessory should rest in the lowest power state that still allows iAP commands to be received and processed.
1	Enable (1) or disable (0) status change notification. When status change notification is disabled, the Apple device assumes that the accessory will send no asynchronous <code>StatusChangeNotify</code> commands. The Apple device may poll the accessory for changes.
2	Reserved
3	Enable RDS/RBDS Raw mode. When enabled, raw RDS/RBDS data format is used by the RDS data commands (commands 0x1A-0x21). When disabled, parsed RDS data is used by the RDS data commands.
7:4	Reserved

Command 0x06: GetTunerBand

Direction: Apple device to Accessory

This command is sent by an Apple device to get an RF tuner accessory's RF band information. The accessory must respond by returning a `RetTunerBand` command.

Table 3-135 GetTunerBand packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x06	Command ID: GetTunerBand
5	0xF1	Checksum

Command 0x07: RetTunerBand

Direction: Accessory to Apple device

This command is sent by an RF tuner accessory to report its tuner band state in response to an Apple device's GetTunerBand command. If the RF tuner current draw from the Apple device is off, it should return its last active band state.

Note: Tuner band state information must be consistent with the accessory's capabilities, as previously reported by a RetTunerCaps command.

Table 3-136 RetTunerBand packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x07	Command ID: RetTunerBand
5	0xNN	Tuner band state information (see Table 3-137 (page 290)):
6	0xNN	Checksum

Table 3-137 Tuner band state IDs

ID	Description
0x00	AM band worldwide (520-1710 KHz)

ID	Description
0x01	Europe/US FM band (87.5–108.0 MHz)
0x02	Japan FM band (76.0–90.0 MHz)
0x03	FM wide band (76.0–108.0 MHz)
0x04–0xFF	Reserved

Command 0x08: SetTunerBand

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory to set its tuner band. In response, the accessory must send the Apple device an ACK command with the command status. The command status must be reported as command failed (command status 0x02) if RF tuner current draw from the Apple device is not on.

Table 3-138 SetTunerBand packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x08	Command ID: SetTunerBand
5	0xNN	Tuner band to be set (see Table 3-137 (page 290))
6	0xNN	Checksum

Note: The tuner band setting requested by the Apple device will be consistent with the RF tuner accessory's capabilities, as previously reported by a RetTunerCaps command.

Command 0x09: GetTunerFreq

Direction: Apple device to Accessory

This command is sent by an Apple device to get an RF tuner accessory's current accessory tuner frequency and signal strength level. In response, the accessory must send the Apple device a RetTunerFreq command.

Table 3-139 GetTunerFreq packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x09	Command ID: GetTunerFreq
5	0xEE	Checksum

Command 0x0A: RetTunerFreq

Direction: Accessory to Apple device

This command is sent by an RF tuner accessory in response to a GetTunerFreq or SetTunerFreq command received from an Apple device. The tuner frequency is expressed in kilohertz: for example, 76000 for 76.0 MHz, 87500 for 87.5 MHz, or 107900 for 107.9 MHz. Valid ranges are 87500 to 107900 for the European/US FM band and 76000 to 89900 for the Japanese FM band.

If the RF tuner accessory's capabilities includes tuner channel RSSI indication, byte 9 of the command packet may be a number greater than zero; otherwise it must be set to 0x00. The tuner RSSI level must be normalized to a value between 0 (minimum) and 255 (maximum). If RF tuner current draw from the Apple device is off, the last active tuner frequency and signal strength (if applicable) should be sent.

IMPORTANT: The accessory must send this command as quickly as possible in response to a GetTunerFreq command from the Apple device. Tuning will fail if the response time is longer than 200 ms.

Table 3-140 RetTunerFreq packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0A	Command ID: RetTunerFreq
5	0xNN	Tuner frequency in kilohertz (bits 31:24)
6	0xNN	Tuner frequency (bits 23:16)
7	0xNN	Tuner frequency (bits 15:8)

Byte number	Value	Comment
8	0xNN	Tuner frequency (bits 7:0)
9	0xNN	Tuner channel received signal strength (<code>rssiLevel</code>) ; may range from 0x00 (no signal present or RSSI indication not supported) to 0xFF (maximum RSSI signal strength).
10	0xNN	Checksum

Note: The tuner channel received signal strength value is valid only if the tuner RSSI bit was set in a previous `RetTunerCaps` command.

Command 0x0B: SetTunerFreq

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory to select a specific tuner frequency within the current tuner band. The tuner frequency is expressed in kilohertz; valid ranges are 87500 to 107900 for the European/US FM band and 76000 to 89900 for the Japanese FM band. In response, the accessory must send a `RetTunerFreq` command with the new tuner frequency and RSSI level (if the accessory supports RSSI). The requested frequency should be within the currently selected tuner band. The command status must be reported as command failed (command status 0x02) if RF tuner current draw from the Apple device is not on.

Table 3-141 SetTunerFreq packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0B	Command ID: SetTunerFreq
5	0xNN	Tuner frequency in kilohertz (bits 31:24)
6	0xNN	Tuner frequency (bits 23:16)
7	0xNN	Tuner frequency (bits 15:8)
8	0xNN	Tuner frequency (bits 7:0)
9	0xNN	Checksum

Command 0x0C: GetTunerMode

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory to get the current tuner mode state from the accessory. In response, the accessory must send the Apple device a `RetTunerMode` command.

Table 3-142 GetTunerMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0C	Command ID: GetTunerMode
5	0xEB	Checksum

Command 0x0D: RetTunerMode

Direction: Accessory to Apple device

This command is sent by an accessory in response to a `GetTunerMode` command received from an Apple device. The tuner mode bits returned are valid only if the associated mode bits returned by a previous `RetTunerCaps` command were set. If tuner power is off, the last active tuner mode information should be returned.

Table 3-143 RetTunerMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0D	Command ID: RetTunerMode
5	0xNN	Tuner mode status (see Table 3-144 (page 295))
6	0xNN	Checksum

Table 3-144 RF Tuner mode status bits

Bits	Description
1:0	FM tuner resolution (see Table 3-129 (page 286))
2	Tuner is currently seeking up or down
3	Tuner is currently seeking with an RSSI minimum threshold enabled
4	Monophonic mode is forced (1) or stereo is allowed (0)
5	Stereo blend is enabled (valid only if bit 4 is 0)
6	FM Tuner deemphasis is 50 μ sec (1) or 75 μ sec (0)
7	AM tuner resolution (1 = 9 KHz, 0 = 10 KHz)

Command 0x0E: SetTunerMode

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory to set the current tuner mode. In response, the accessory must send an RF tuner lingo ACK command with the command status. The command status must be reported as command failed (command status 0x02) if RF tuner current draw from the Apple device is not on.

Table 3-145 SetTunerMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0E	Command ID: SetTunerMode
5	0xNN	Tuner mode to be set (see Table 3-146 (page 295))
6	0xNN	Checksum

Table 3-146 Set RF Tuner mode bits

Bits	Description
1:0	Set FM tuner resolution (see Table 3-129 (page 286))
2-3	Reserved; set to 0

Bits	Description
4	Force monophonic mode (1) or allow stereo (0)
5	Enable stereo blend (valid only if bit 4 is 0); this blends the source left and right channels to improve the perceived signal quality while still retaining some stereo image separation
6	Set FM Tuner deemphasis to 50 μ sec (1) or 75 μ sec (0)
7	Set AM tuner resolution (1 = 9 KHz, 0 = 10 KHz)

Note: `SetTunerMode` must not be used to start tuner seeking or set the seek type. The `TunerSeekStart` command must be used instead.

Command 0x0F: GetTunerSeekRssi

Direction: Apple device to Accessory

This command is sent by an Apple device to get the current tuner seek threshold value from an RF tuner accessory. In response, the accessory must send a `RetTunerSeekRssi` command with the seek RSSI threshold, if supported by the accessory. If seek RSSI is not supported, the RF tuner accessory must return an `ACK` command with failure status.

Table 3-147 GetTunerSeekRssi packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0F	Command ID: GetTunerSeekRssi
5	0xE8	Checksum

Command 0x10: RetTunerSeekRssi

Direction: Accessory to Apple device

This command is sent by an accessory in response to a `GetTunerSeekRssi` command received from an Apple device, assuming the accessory supports the seek RSSI capability. Its threshold value represents the minimum signal strength that allows a tuner channel to be recognized during the seek process; it is normalized to a value between 0 (minimum) and 255 (maximum). If current draw from the Apple device is off, the last active RSSI threshold value should be sent.

Table 3-148 RetTunerSeekRssi packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x10	Command ID: RetTunerSeekRssi
5	0xNN	RSSI threshold for seeking action
6	0xNN	Checksum

Command 0x11: SetTunerSeekRssi

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory that supports the seek RSSI capability, to set the accessory's seek RSSI signal strength threshold. The threshold value is the minimum signal strength that allows a tuner channel to be recognized during the seek process; it is normalized to a range between 0 (minimum) and 255 (maximum).

Table 3-149 SetTunerSeekRssi packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x11	Command ID: SetTunerSeekRssi
5	0xNN	RSSI threshold for seeking action
6	0xNN	Checksum

Command 0x12: TunerSeekStart

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory that supports the tuner seek up/down capability (as reported by a previous RetTunerCaps command), to initiate seeking of a specified type. The returned ACK command status must be reported as command failed (command status 0x02) if RF tuner

current draw from the Apple device is not on. Seeking operations that use an RSSI threshold are initiated only if the RF tuner accessory's seek RSSI threshold capability is also supported, as reported by the `RetTunerCaps` command.

Table 3-150 TunerSeekStart packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x12	Command ID: TunerSeekStart
5	0xNN	ID of tuner seeking operation (see Table 3-151 (page 298) and Note below)
6	0xNN	Checksum

Note: An accessory that declares support for HD radio by setting its capability bit 04 (see [Table 3-128](#) (page 285)) must also support HD seeks. If the accessory receives a seek code it does not support, it must return a `DevACK` command with Bad Parameter status (0x04).

Table 3-151 Tuner seeking operations

ID	Operation	Use RSSI threshold
0x00	No seek operation (cancel seek operation, if active)	N/A
0x01	Seek up from beginning of band	No
0x02	Seek down from end of band	No
0x03	Seek up from current frequency	No
0x04	Seek down from current frequency	No
0x05	Seek up from beginning of band	Yes
0x06	Seek down from end of band	Yes
0x07	Seek up from current frequency	Yes
0x08	Seek down from current frequency	Yes
0x09	Seek up from beginning of band for an HD signal	No
0x0A	Seek down from end of band for an HD signal	No
0x0B	Seek up from current frequency for an HD signal	No

ID	Operation	Use RSSI threshold
0x0C	Seek down from current frequency for an HD signal	No
0x0D	Seek up from beginning of band for an HD signal	Yes
0x0E	Seek down from end of band for an HD signal	Yes
0x0F	Seek up from current frequency for an HD signal	Yes
0x10	Seek down from current frequency for an HD signal	Yes
0x11–0xFF	Reserved	N/A

An analog seek operation (operation 0x01–0x08) will seek any frequency that contains a valid signal including HD signals. An HD seek operation (operation 0x09–0x10) skips channels that are not HD and stops only on channels with HD signals present. The HD seek completes when either of two conditions is satisfied:

- An HD channel that satisfies the criteria of the tuner's seek function was located within the band. This may result in moving one or more channel spacings and wrapping around one end of the band.
- No HD channel that satisfies the criteria of the tuner's seek function was located within the band, and the seek has traversed the entire band and wrapped back to the original tuner frequency.

A seek operation using an RSSI threshold completes when either of two conditions is satisfied:

- A channel was located within the band that satisfies the minimum RSSI threshold level.
- No channel was located within the band that satisfies the minimum the RSSI threshold level. The seek has traversed the entire band and wrapped back to the beginning tuner frequency without locating a valid channel. If no channel is found, it may indicate that the threshold is too high for the current radio reception area.

A seek operation using no RSSI threshold completes when either of two conditions is satisfied:

- A channel was located within the band that satisfies the criteria of the tuner's seek function. This may result in moving one or more channel spacings and wrapping around at the band ends.
- No channel was located within the band that satisfies the criteria of the tuner's seek function and the seek has traversed the entire band and wrapped back to the beginning tuner frequency without locating a valid channel.

An **ACK** command with command pending status is returned for seek operations requiring more than 100 ms to complete. It indicates the maximum time required for the accessory to complete the requested scan type. When the requested seek operation is completed (either successfully or unsuccessfully), the accessory must respond with a **TunerSeekDone** command indicating the seek operation status. If the accessory does not support tuner seek (with RSSI) operations or if tuner power is off, an **ACK** command with error status is returned.

When a cancel seek operation is requested and a seek operation is active, the seek operation stops at the current seek channel and the accessory responds with a **TunerSeekDone** command indicating the frequency and RSSI of the channel.

Command 0x13: TunerSeekDone

Direction: Accessory to Apple device

In response to a `TunerSeekStart` command from an Apple device, an RF tuner accessory must send this command to the Apple device after the seek operation has finished. It reports the current tuner frequency, which is assumed to be the result of the seek operation. If the accessory supports a tuner channel RSSI indication capability (as reported by a previous `RetTunerCaps` command), the `rssiLevel` value shows the current channel's RSSI signal strength level. If no channel was found, a tuner frequency value of `0xFFFFFFFF` must be reported. The received signal strength value must be normalized to a range from `0x00` (no signal or the accessory does not support RSSI indication) to `0xFF` (maximum signal strength).

If an HD signal seek was requested, the tuner should tune to the next frequency containing HD content but it should not select an analog or HD program service. The Apple device will perform the program selection operation separately. See "[Command 0x28: RetHDPProgramService](#)" (page 315) and "[Command 0x29: SetHDPProgramService](#)" (page 316).

Table 3-152 TunerSeekDone packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x13	Command ID: TunerSeekDone
5	0xNN	Tuner frequency in kilohertz (bits 31:24)
6	0xNN	Tuner frequency in kilohertz (bits 23:16)
7	0xNN	Tuner frequency in kilohertz (bits 15:8)
8	0xNN	Tuner frequency in kilohertz (bits 7:0)
9	0xNN	Tuner channel RSSI received signal strength value
10	0xNN	Checksum

Command 0x14: GetTunerStatus

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory to get its current tuner status state; in response, the accessory must send a `RetTunerStatus` command. The command can be used to poll the accessory's status if the accessory does not support status change notifications. After the accessory's status is reported its status bits must be cleared, so that an immediate reread will return no active status.

Table 3-153 GetTunerStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x14	Command ID: GetTunerStatus
5	0xE3	Checksum

Command 0x15: RetTunerStatus

Direction: Accessory to Apple device

This command is sent by an RF tuner accessory in response to a GetTunerStatus command received from an Apple device. The tuner status bits returned are valid only if the status capability bits returned by a previous RetTunerCaps command were set. If the tuner power is off, the last active tuner status information should be returned.

Table 3-154 RetTunerStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x15	Command ID: RetTunerStatus
5	0xNN	Tuner status bits (see Table 3-155 (page 301))
6	0xNN	Checksum

Note: An accessory that declares support for HD radio by setting its capability bit 04 must return valid RF tuner status bits; see [Table 3-128](#) (page 285).

Table 3-155 RF tuner status bits

Bit	Description
0	RDS/RBDS data is received, ready to read

Bit	Description
1	Tuner channel RSSI level has changed
2	Stereo source indicator state; 1 for a stereo signal source, 0 for monophonic
3	HD signal present
4	HD digital audio present
5	HD data is received; ready to read
7:6	Reserved

Command 0x16: GetStatusNotifyMask

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory to get the status notification mask from the accessory. This mask indicates which state changes will invoke a notification change command from the accessory. In response, the accessory must send the Apple device a `RetStatusNotifyMask` command.

Table 3-156 GetStatusNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x16	Command ID: GetStatusNotifyMask
5	0xE1	Checksum

Command 0x17: RetStatusNotifyMask

Direction: Accessory to Apple device

This command must be returned by the accessory in response to the `GetStatusNotifyMask` command. The status notification mask indicates which state changes will invoke a notification change command from the accessory. However, its bit values are valid only if the corresponding capabilities bits returned by a previous `RetTunerCaps` command were set. A mask bit value of 1 indicates that notification is enabled; a value of 0 indicates notification is disabled or not supported.

Table 3-157 RetStatusNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x17	Command ID: RetStatusNotifyMask
5	0xNN	Status notification mask (see Table 3-158 (page 303))
6	0xNN	Checksum

Note: An accessory that declares support for HD radio by setting its capability bit 04 (see [Table 3-128](#) (page 285)) must support status notifications.

Table 3-158 Status notification mask bits

Bit	Description
0	RDS/RBDS data-ready change notify enabled (ignored if SetRdsNotifyMask sets rdsMask to a value other than zero)
1	Tuner channel RSSI-level change notify enabled
2	Stereo indicator state change notify enabled
3	HD signal present notification enabled
4	HD digital audio present notification enabled
5	HD data ready notification enabled
7:6	Reserved

Command 0x18: SetStatusNotifyMask

Direction: Apple device to Accessory

The Apple device sends this command to an RF tuner accessory to set the status change notification mask. The status notification mask indicates which state changes will invoke a notification change command from the accessory. However, its bit values are valid only if the corresponding capabilities bits returned by a previous RetTunerCaps command were set. A mask bit value of 1 indicates that notification is enabled; a value of 0 indicates notification is disabled or not supported.

Note: For RDS/RBDS data-ready changes, the `SetRdsNotifyMask` command can override the notification status mask bit. If the `SetRdsNotifyMask` notification mask is set with a nonzero mask, then any enabled RDS/RBDS data notifications will be sent using the `RdsReadyNotify` command instead of the `StatusChangeNotify` command. For HD data ready changes, the `SetHDDDataNotifyMask` command can override the HD data notification status mask bit. If the `SetHDDDataNotifyMask` notification mask is set with a nonzero mask, then any enabled HD data notifications must be sent using the `HDDDataReadyNotify` command instead of the `StatusChangeNotify` command.

Table 3-159 `SetStatusNotifyMask` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x18	Command ID: <code>SetStatusNotifyMask</code>
5	0xNN	Status notification mask (see Table 3-160 (page 304))
6	0xNN	Checksum

Table 3-160 Status notification mask setting bits

Bit	Description
0	Enable RDS/RBDS data-ready change notification (ignored if <code>SetRdsNotifyMask</code> sets <code>rdsMask</code> to a value other than zero)
1	Enable tuner channel RSSI-level change notification
2	Enable stereo-indicator state change notification
3	Enable HD signal present notification
4	Enable HD digital audio present notification
5	Enable HD data ready notification
7:6	Reserved

Command 0x19: `StatusChangeNotify`

Direction: Accessory to Apple device

This command must be sent asynchronously by an RF tuner accessory to an attached Apple device to report each enabled status change. The Apple device enables specific RF tuner status change notifications using the `SetStatusNotifyMask` command. After a notification has been sent, the status bits should be automatically cleared so they are ready to receive the next status change.

Note: Tuner channel RSSI-level change and stereo-indicator state change notifications must not occur more than once every 100 ms.

Table 3-161 StatusChangeNotify packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x19	Command ID: StatusChangeNotify
5	0xNN	Status change ID bits (see Table 3-162 (page 305))
6	0xNN	Checksum

Note: For RDS/RBDS data-ready changes, the `SetRdsNotifyMask` command can override the notification status mask bit. If the `SetRdsNotifyMask` notification mask is set with a nonzero mask, then any enabled RDS/RBDS data notifications will be sent using the `RdsReadyNotify` command instead of the `StatusChangeNotify` command. Similarly, for HD data-ready changes, the `SetHDDDataNotifyMask` command can override the HD data notification status mask bit. If the `SetHDDDataNotifyMask` notification mask is set with a nonzero mask, then any enabled HD data notifications will be sent using the `HDDDataReadyNotify` command instead of the `StatusChangeNotify` command.

Table 3-162 Status change ID bits

Bit	Description
0	RDS/RBDS data-ready change (valid only if bit 0 of <code>statusMask</code> is 1 and <code>SetRdsNotifyMask</code> sets <code>rdsMask</code> to 0x0)
1	Tuner channel RSSI-level change
2	Stereo indicator state change
3	HD signal present
4	HD digital audio present
5	HD data is received; ready to read
7:6	Reserved

Command 0x1A: GetRdsReadyStatus

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory to get the accessory's current RDS/RBDS data-ready status. It can be used to poll the accessory's RDS/RBDS data-ready status without having to enable RDS/RBDS data-ready notifications. This command is usable only if the `RetTunerCaps` capability bits report has indicated that the accessory supports RDS/RBDS data reception and parsing. In response, the accessory must send a `RetRdsReadyStatus` command.

Table 3-163 GetRdsReadyStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1A	Command ID: GetRdsReadyStatus
5	0xDD	Checksum

Command 0x1B: RetRdsReadyStatus

Direction: Accessory to Apple device

This command must be sent by an RF tuner accessory in response to a `GetRdsReadyStatus` command received from an Apple device. Its status value indicates which RDS/RBDS data values are available to be read. All status bits must remain set on the accessory until the Apple device sends a `GetRdsData` command to read the data. If the accessory does not support RDS/RBDS data, it must send an `rdsReady` data-ready status of 0 to indicate that no data is ready.

Table 3-164 RetRdsReadyStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1B	Command ID: RetRdsReadyStatus
5	0xNN	RDS/RBDS data-ready status (bits 31:24) (see Table 3-165 (page 307) and Table 3-166 (page 307))

Byte number	Value	Comment
6	0xNN	RDS/RBDS data-ready status (bits 23:16)
7	0xNN	RDS/RBDS data-ready status (bits 15:8)
8	0xNN	RDS/RBDS data-ready status (bits 7:0)
9	0xNN	Checksum

Table 3-165 RDS/RBDS data-ready status bits, parsed mode

Bit	Description
03-00	Reserved; set to zeros
04	RadioText (RT) data-ready
29-05	Reserved; set to zeros
30	Program Service Name (PSN) data-ready
31	Reserved; set to zero

Table 3-166 RDS/RBDS data-ready status bits, raw mode

Bit	Description
04-00	Reserved
05	RDS/RBDS group data ready
31-06	Reserved

Command 0x1C: GetRdsData

Direction: Apple device to Accessory

This command is sent by an Apple device to get raw or unparsed RDS/RBDS data from an RF tuner accessory that supports the RDS/RBDS data capability. If the accessory supports RDS/RBDS and has data-ready, it must send a `RetRdsData` command. If the accessory does not support RDS/RBDS or does not have the specified data type ready, it must return an `ACK` command with command failure status.

Table 3-167 `GetRdsData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1C	Command ID: GetRdsData
5	0xNN	RDS/RBDS data type (<code>rdsDataType</code>) ID (see Table 3-168 (page 308) and Table 3-169 (page 308))
6	0xNN	Checksum

Table 3-168 RDS/RBDS data type IDs, parsed mode

ID Bytes	Description
0x00–0x03	Reserved
0x04	RadioText (RT)
0x05–0x1D	Reserved
0x1E	Program Service Name (PSN)
0x1F–0xFF	Reserved

Table 3-169 RDS/RBDS data type IDs, raw mode

ID Bytes	Description
0x00–0x04	Reserved
0x05	RDS/RBDS group data ready
0x06–0xFF	Reserved

Command 0x1D: RetRdsData

Direction: Accessory to Apple device

This command is sent by an RF tuner accessory that has an RDS/RBDS data group ready in response to a `GetRdsData` command received from an Apple device. If the accessory does not support the RDS/RBDS capability or does not have the specified data type ready, it must return an `ACK` command with command failure status (command status 0x02).

Table 3-170 `RetRdsData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1D	Command ID: RetRdsData
5	0xNN	rdsDataType (see Table 3-171 (page 309))
6	0xNN	rdsData (see Table 3-171 (page 309))
NN	0xNN	Checksum

Table 3-171 rdsDataType bytes and rdsData formats

rdsDataType bytes	Description	rdsData format
0x00–0x03	Reserved	N/A
0x04	RadioText (RT)	charSet:1 (see Table 3-172 (page 309)), radioText:64
0x05–0x1D	Reserved	N/A
0x1E	Program Service Name (PSN)	charSet:1 (see Table 3-172 (page 309)), progSvcName:8
0x1F–0xFF	Reserved	N/A

Note: The maximum length of a Radio Text is 64 bytes and that of a Program Service Name is 8 bytes.

Table 3-172 rdsData character set IDs

ID	Character set
0x00	Latin-based languages
0x01	Cyrillic- and Greek-based languages
0x02	Arabic- and Hebrew-based languages
0x03–0xFF	Reserved

When RDS/RBDS Raw mode is enabled, the data shown in [Table 3-173](#) (page 310) is returned for the rdsDataType x05 (RDS/RBDS group data ready).

Table 3-173 RDS/RBDS group data-ready data

Byte	Description
0-1	16 bit value for Block A of RDS/RBDS group data
2-3	16 bit value for Block B of RDS/RBDS group data
4-5	16 bit value for Block C of RDS/RBDS group data
6-7	16 bit value for Block D of RDS/RBDS group data
8	Block errors byte (see Table 3-174 (page 310))

Table 3-174 Block errors byte encoding

Byte	Description	
1:0	Block A error bits	See Table 3-175 (page 310)
3:2	Block B error bits	
5:4	Block C error bits	
7:6	Block D error bits	

Table 3-175 Block error bit values

Bit values	Description
0,0	No errors
0,1	1 to 2 errors; slight chance of uncorrected errors
1,0	3 to 5 errors; may have uncorrected errors
1,1	6 or more errors; uncorrectable block

Command 0x1E: GetRdsNotifyMask

Direction: Apple device to Accessory

This command is sent by an Apple device to get an RF tuner accessory's current RDS/RBDS data notification mask (`rdsMask`). In response, the accessory must send a `RetRdsNotifyMask` command with the RDS/RBDS data notification mask. This command is valid only if the RDS/RBDS data-ready capability bit was set in a previous `RetTunerCaps` command.

Table 3-176 GetRdsNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1E	Command ID: GetRdsNotifyMask
5	0xD9	Checksum

Command 0x1F: RetRdsNotifyMask

Direction: Accessory to Apple device

This command is sent by an RF tuner accessory in response to a GetRdsNotifyMask command sent by an Apple device. The RDS/RBDS mask (`rdsMask`) indicates which asynchronous data notifications should be sent by the accessory when data is ready. For each data type bit, 1 means that data-ready notification is enabled and 0 means that notification is disabled. For enabled RDS/RBDS data types, the accessory must send RdsReadyNotify commands to the Apple device when the associated data becomes available.

Table 3-177 RetRdsNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1F	Command ID: RetRdsNotifyMask
5	0xNN	RDS/RBDS data change notification mask (bits 31:24) (see Table 3-178 (page 312) and Table 3-179 (page 312))
6	0xNN	RDS/RBDS data change notification mask (bits 23:16)
7	0xNN	RDS/RBDS data change notification mask (bits 15:8)
8	0xNN	RDS/RBDS data change notification mask (bits 7:0)
9	0xNN	Checksum

Table 3-178 RDS/RBDS data change notification mask bits, parsed mode

Bit	Description
03-00	Reserved
04	RadioText (RT)
29-05	Reserved (must be set to zeros)
30	Program Service Name (PSN)
31	Reserved (must be set to 0)

Table 3-179 RDS/RBDS data change notification mask bits, raw mode

Bit	Description
04-00	Reserved
05	RDS/RBDS group data
31-06	Reserved

Command 0x20: SetRdsNotifyMask

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory to set the accessory's RDS/RBDS data-ready notification mask (`rdsMask`). When notification bits are enabled, the accessory must send an `RdsReadyNotify` command with the associated RDS/RBDS data. For each RDS/RBDS data type bit, 1 enables data-ready notification and 0 disables notification.

Note: For RDS/RBDS data-ready changes, the `SetRdsNotifyMask` command can override the notification status mask bit. If the `SetRdsNotifyMask` notification mask is set with a nonzero mask, then any enabled RDS/RBDS data notifications will be sent using the `RdsReadyNotify` command instead of the `StatusChangeNotify` command.

Table 3-180 `SetRdsNotifyMask` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x20	Command ID: <code>SetRdsNotifyMask</code>

Byte number	Value	Comment
5	0xNN	RDS/RBDS data change notification mask (bits 31:24) (see Table 3-181 (page 313) and Table 3-182 (page 313))
6	0xNN	RDS/RBDS data change notification mask (bits 23:16)
7	0xNN	RDS/RBDS data change notification mask (bits 15:8)
8	0xNN	RDS/RBDS data change notification mask (bits 7:0)
9	0xNN	Checksum

Table 3-181 RDS/RBDS data change notification mask setting bits, parsed mode

Bit	Description
03-00	Reserved
04	RadioText (RT)
29-05	Reserved
30	Program Service Name (PSN)
31	Reserved

Table 3-182 RDS/RBDS data change notification mask setting bits, raw mode

Bit	Description
04-00	Reserved
05	RDS/RBDS group data
31-06	Reserved

Command 0x21: RdsReadyNotify

Direction: Accessory to Apple device

This command must be sent by an RF tuner accessory when RDS/RBDS data is ready, the associated `SetRdsNotifyMask` bit is set, and the accessory's capability bit from a previous `RetTunerCaps` command indicates that the accessory supports RDS/RBDS status notifications. After a notification command is sent, the accessory's associated RDS/RBDS data-ready status bit must be cleared.

Table 3-183 `RdsReadyNotify` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x21	Command ID: RdsReadyNotify
5	0xNN	rdsDataType (for RDS data types, see Table 3-171 (page 309))
6	0xNN	rdsData (for RDS data types, see Table 3-171 (page 309))
NN	0xNN	Checksum

Command 0x25: GetHDProgramServiceCount

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF Tuner accessory to get the count of HD program services broadcast at the current tuner frequency. In response, the accessory must send a `RetHDProgramServiceCount` command with the count of HD program services available. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

Table 3-184 GetHDProgramServiceCount packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x25	Command ID: GetHDProgramServiceCount
5	0xD2	Checksum

Command 0x26: RetHDProgramServiceCount

Direction: Accessory to Apple device

This command is sent by an RF Tuner accessory in response to a `GetHDProgramServiceCount` command sent by an Apple device. The command returns the count of HD program services broadcast at the current tuner frequency. The command returns a count of 0 if there are no HD program services available. HD programs services must be indexed from 1 to the count value, 1 being the main program service. The Analog Program Exists field indicates whether the station has analog programming or not. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

Table 3-185 RetHDProgramServiceCount packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x26	Command ID: RetHDProgramServiceCount
5	0xNN	HD program service count (0-8)
6	0xNN	Analog Program Exists (0 = no, 1 = yes)
7	0xNN	Checksum

Command 0x27: GetHDProgramService

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF Tuner accessory to get the current tuned HD program service. In response, the accessory must send a RetHDProgramService command with the current HD program service. This command is only valid if the HD capable bit was set in a previous RetTunerCaps command.

Table 3-186 GetHDProgramService packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x27	Command ID: GetHDProgramService
5	0xD0	Checksum

Command 0x28: RetHDProgramService

Direction: Accessory to Apple device

This command is sent by an RF Tuner accessory in response to a `GetHDProgramService` command sent by an Apple device. The value returned is the tuned HD program service, 0 if the analog program is currently tuned, or 0xFF if audio decoding and output is currently disabled. The output may be disabled if a `SetHDProgramService` command was sent with parameter 0xFF or if the radio has just been tuned to a station's frequency. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

Table 3-187 RetHDProgramService packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x28	Command ID: RetHDProgramService
5	0xNN	HD program service index (0x00-0x08 or 0xFF)
6	0xNN	Checksum

Command 0x29: SetHDProgramService

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF Tuner accessory to tune to an HD program service. Tuning to program service 0 disables HD tuning and switches back to analog tuning, if it is available. Tuning to program service 0xFF disables all audio decoding and output. This command lets the Apple device retrieve information for all available HD programs before selecting a program's audio to be decoded and presented. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

Selecting an analog program on an HD digital-only station or attempting to select a nonexistent HD program constitutes an invalid program selection. In this case, the accessory should send the Apple device an `ACK` command with Bad Parameter status.

Table 3-188 SetHDProgramService packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x29	Command ID: SetHDProgramService

Byte number	Value	Comment
5	0xNN	HD program service index (0x00-0x08 or 0xFF)
6	0xNN	Checksum

Command 0x2A: GetHDDataReadyStatus

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory to get the accessory's current HD data-ready status. It can be used to poll the accessory's HD data-ready status without having to enable HD data-ready notifications. In response, the accessory must send a `RetHDDataReadyStatus` command. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

Note: Only parsed mode is supported for HD Data.

Table 3-189 GetHDDataReadyStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2A	Command ID: GetHDDataReadyStatus
5	0xCD	Checksum

Command 0x2B: RetHDDataReadyStatus

Direction: Accessory to Apple device

This command must be sent by an RF tuner accessory in response to a `GetHDDataReadyStatus` command received from an Apple device. Its status value indicates which HD data values are available to be read. All status bits must remain set on the accessory until the Apple device sends a `GetHDData` command to read the data.

Table 3-190 RetHDDataReadyStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2B	Command ID: RetHDDataReadyStatus
5	0xNN	HD data-ready status (bits 31:24) (see Table 3-191 (page 318))
6	0xNN	HD data-ready status (bits 23:16)
7	0xNN	HD data-ready status (bits 15:8)
8	0xNN	HD data-ready status (bits 7:0)
9	0xNN	Checksum

Table 3-191 HD data-ready status bits

Bit	Description
0	PSD data ready
1	Reserved
2	SIS Station ID number data-ready
3	SIS Station Name (short) data-ready
4	SIS Station Name (long) data-ready
5	SIS ALFN data-ready
6	SIS Station Location data-ready
7	SIS Station Message data-ready
8	SIS Slogan data-ready
9	SIS Parameter Message data-ready
31:10	Reserved

Command 0x2C: GetHDData

Direction: Apple device to Accessory

This command is sent by an Apple device to get HD data from an RF tuner accessory. If the accessory supports HD and has data-ready, it must send a RetHDData command. If the accessory does not support HD or does not have the specified data type ready, it must return an ACK command with command failure status. This command is valid only if the HD capable bit was set in a previous RetTunerCaps command.

Note: Only parsed mode is supported for HD Data.

Table 3-192 GetHDData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2C	Command ID: GetHDData
5	0xNN	HD data type; see Table 3-193 (page 319)
6	0xNN	Checksum

Table 3-193 HD data type IDs

Type ID	Description
0x00	PSD data
0x01	Reserved
0x02	SIS Station ID number data
0x03	SIS Station Name (short) data
0x04	SIS Station Name (long) data
0x05	SIS ALFN data
0x06	SIS Station Location data
0x07	SIS Station Message data
0x08	SIS Slogan
0x09	SIS Parameter Message data
0x0A-0xFF	Reserved

Command 0x2D: RetHDData

Direction: Accessory to Apple device

This command is sent by an RF Tuner accessory that has HD data ready in response to a `GetHDDData` command. If the accessory does not have the specified data type ready, it must return an `ACK` command with command failure status (command status 0x02). On sending this command, the accessory must keep clear its internal HD data-ready status for the data type being read until the next time data is ready.

Table 3-194 RetHDDData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2D	Command ID: RetHDDData
5	0xNN	HDDDataType (see Table 3-195 (page 320))
6	0xNN	HDDData
NN	0xNN	Checksum

Table 3-195 HDDDataType type IDs and formats

HDDDataType ID	HDDData format
0x00	8 Bit integer index, 8 Bit integer max index, 8 bit HD program index, ID3 Frame data (see Table 3-196 (page 321))
0x01	N/A
0x02	32-bit integer station ID
0x03	8-bit character encoding type, up to 12 bytes (4 or 7 in USA) of short station name (see Table 3-197 (page 321))
0x04	0-56 characters station name (UTF-7)
0x05	32-bit ALFN
0x06	27-bit GPS latitude/longitude encoded in lower bits of 32 bit data
0x07	8-bit character encoding type, 0-190 bytes station message (see Table 3-197 (page 321))
0x08	8-bit character encoding type, 2-96 bytes station slogan (see Table 3-197 (page 321))
0x09	6-bit index encoded in lower bits of 8 bit data, 16-bit parameter
0x0A–0xFF	N/A

Table 3-196 PSD data format

HDDDataType ID	Description
0x00	Current ID3 Frame index (0 = first)
0x01	Last Index of ID3 Frame data
0x02	HD Program Index (1-8)
0x03–0xNN	ID3 Frame data, ID3 tag version 2.3.0. The first byte of the ID3 tag's text information contains the encoding type.

Table 3-197 8-bit character encoding type formats

Encoding type ID	Description
0x00	ISO/IEC 8859-1:1998
0x01–0x03	Reserved
0x04	ISO/IEC 10646-1:2000, UCS-2 (Little-endian)
0x05–0xFF	Reserved

Command 0x2E: GetHDDDataNotifyMask

Direction: Apple device to Accessory

This command is sent by an Apple device to get an RF tuner accessory's current HD data notification mask. In response, the accessory must send a `RetHDDDataNotifyMask` command with the HD data notification mask. This command is valid only if the HD capability bit was set in a previous `RetTunerCaps` command.

Table 3-198 GetHDDDataNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2E	Command ID: GetHDDDataNotifyMask
5	0xC9	Checksum

Command 0x2F: RetHDDataNotifyMask

Direction: Accessory to Apple device

This command is sent by an RF tuner accessory in response to a `GetHDDataNotifyMask` command sent by an Apple device. The HD mask indicates which asynchronous data notifications should be sent by the accessory when data is ready. For each data type bit, 1 means that data-ready notification is enabled and 0 means that notification is disabled. For enabled HD data types, the accessory must send `HDDataReadyNotify` commands to the Apple device when the associated data becomes available.

Table 3-199 RetHDDataNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2F	Command ID: RetHDDataNotifyMask
5	0xNN	HD data change notification mask (bits 31:24) (see Table 3-200 (page 322))
6	0xNN	HD data change notification mask (bits 23:16)
7	0xNN	HD data change notification mask (bits 15:8)
8	0xNN	HD data change notification mask (bits 7:0)
9	0xNN	Checksum

Table 3-200 HD data change notification mask bits

Bit	Description
0	PSD data ready
1	Reserved
2	SIS Station ID number data-ready
3	SIS Station Name (short) data-ready
4	SIS Station Name (long) data-ready
5	SIS ALFN data-ready
6	SIS Station Location data-ready
7	SIS Station Message data-ready

Bit	Description
8	SIS Slogan data-ready
9	SIS Parameter Message data-ready
31:10	Reserved

Command 0x30: SetHDDDataNotifyMask

Direction: Apple device to Accessory

This command is sent by an Apple device to an RF tuner accessory to set the accessory's HD data-ready notification mask. When notification bits are enabled, the accessory must send an `HDDataReadyNotify` command with the associated HD data. For each HD data type bit, 1 enables data-ready notification and 0 disables notification. This command is valid only if the HD capability bit was set in a previous `RetTunerCaps` command.

Table 3-201 SetHDDDataNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x30	Command ID: SetHDDDataNotifyMask
5	0xNN	HD data change notification mask (bits 31:24) (see Table 3-202 (page 323))
6	0xNN	HD data change notification mask (bits 23:16)
7	0xNN	HD data change notification mask (bits 15:8)
8	0xNN	HD data change notification mask (bits 7:0)
9	0xNN	Checksum

Table 3-202 HD data change notification mask setting bits

Bit	Description
0	PSD data ready
1	Reserved
2	SIS Station ID number data-ready

Bit	Description
3	SIS Station Name (short) data-ready
4	SIS Station Name (long) data-ready
5	SIS ALFN data-ready
6	SIS Station Location data-ready
7	SIS Station Message data-ready
8	SIS Slogan data-ready
9	SIS Parameter Message data-ready
31:10	Reserved

Command 0x31: HDDataReadyNotify

Direction: Accessory to Apple device

This command must be sent by an RF tuner accessory when HD data is ready, the associated `SetHDDataNotifyMask` bit is set, and the accessory's capability bit from a previous `RetTunerCaps` command indicates that the accessory supports HD status notifications. After a notification command is sent, the accessory's associated HD data-ready status bit must be cleared.

Table 3-203 HDDataReadyNotify packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x31	Command ID: HDDataReadyNotify
5	0xNN	HDDataType (see Table 3-204 (page 324))
6–0xNN	0xNN	HD data (see Table 3-205 (page 325))
NN	0xNN	Checksum

Table 3-204 HD data types

Type ID	Description
0x00	PSD data

Type ID	Description
0x01	Reserved
0x02	SIS Station ID number data
0x03	SIS Station Name (short) data
0x04	SIS Station Name (long) data
0x05	SIS ALFN data
0x06	SIS Station Location data
0x07	SIS Station Message data
0x08	SIS Slogan data
0x09	SIS Parameter Message data
0x0A–0xFF	Reserved

Table 3-205 HD Data Type type IDs and formats

HD data type ID	HDData format
0x00	8 Bit integer index, 8 Bit integer max index, 8 bit HD program index, ID3 Frame data (see Table 3-206 (page 326))
0x01	N/A
0x02	32-bit integer station ID
0x03	8-bit character encoding type, up to 12 bytes (4 or 7 in USA) of short station name (see Table 3-207 (page 326))
0x04	0-56 characters station name (UTF-7)
0x05	32-bit ALFN
0x06	27-bit GPS latitude/longitude encoded in lower bits of 32 bit data
0x07	8-bit character encoding type, 0-190 bytes station message (see Table 3-207 (page 326))
0x08	8-bit character encoding type, 2-96 bytes station slogan (see Table 3-207 (page 326))
0x09	6-bit index encoded in lower bits of 8 bit data, 16-bit parameter
0x0A–0xFF	N/A

Table 3-206 PSD data format

HDDataType ID	Description
0x00	Current ID3 Frame index (0 = first)
0x01	Last Index of ID3 Frame data
0x02	HD Program Index (1-8)
0x03–0xNN	ID3 Frame data, ID3 tag version 2.3.0. The first byte of the ID3 tag's text information contains the encoding type.

Table 3-207 8-bit character encoding type formats

Encoding type ID	Description
0x00	ISO/IEC 8859-1:1998
0x01–0x03	Reserved
0x04	ISO/IEC 10646-1:2000, UCS-2 (Little-endian)
0x05–0xFF	Reserved

Sample HD Command Sequences

This section presents three typical examples of using RF Tuner lingo commands with HD radio.

Initializing HD Radio

This example sets up HD Radio using RF Tuner lingo commands.

Table 3-208 Example of HD radio setup

Command	Direction	Data	Comments
GetTunerCaps	Dev to Acc	No Data	Request RF Tuner accessory's supported capabilities.
RetTunerCaps	Acc to Dev	0x07FC0712	Return 32-bit field indicating the accessory's capabilities (FM band US, HD Radio, Tuner power control, status change notification, FM resolution 200 kHz, Tuner seek capable, Tuner seek RSSI threshold capable, Force monophonic mode capable, Stereo blend capable, FM Tuner deemphasis select capable, AM resolution 10 kHz, RDS/RBDS data capable, Tuner channel RSSI indication capable, Stereo source indicator capable).
SetTunerCtrl	Dev to Acc	0x01	Tuner power on, status change notification off, raw mode off.

Command	Direction	Data	Comments
ACK	Acc to Dev	0x00	Acknowledge SetTunerCtrl
SetStatusNotifyMask	Dev to Acc	0x38	Set status notification for HD signal present, HD digital audio present, HD data ready.
ACK	Acc to Dev	0x00	Acknowledge SetStatusNotifyMask
SetHDDDataNotifyMask	Dev to Acc	0x00000009	Set HD data ready notification for Station Short Name and PSD Data.
ACK	Acc to Dev	0x00	Acknowledge SetHDDDataNotifyMask
SetTunerMode	Dev to Acc	0x00	Set FM Tuner resolution 200kHz, stereo allowed, no stereo blend, FM Tuner de-emphasis 75 μ sec, AM Tuner resolution 10kHz.
ACK	Acc to Dev	0x00	Acknowledge SetTunerMode

HD Radio Tuning and Reception

Using RF Tuner lingo commands, this example tunes to an HD radio station, collects data on it, and responds to a loss of digital audio.

Table 3-209 Example of HD radio tuning and reception

Command	Direction	Data	Comments
SetTunerBand	Dev to Acc	0x01	Set Tuner band to FM US.
ACK	Acc to Dev	0x00	Acknowledge SetTunerBand
SetTunerFreq	Dev to Acc	97700	Set FM Tuner frequency to 97.7 MHz
RetTunerFreq	Acc to Dev	97700, 31	Return FM Tuner frequency set to 97.7 MHz with RSSI level of 31.
SetTunerSeekRssi	Dev to Acc	8	Set the tuner seek RSSI threshold to 8.
ACK	Acc to Dev	0x00	Acknowledge SetTunerSeekRssi
TunerSeekStart	Dev to Acc	0x0F	Start tuner seek up from current frequency while checking for RSSI threshold.
ACK	Acc to Dev	0x00	Acknowledge TunerSeekStart
TunerSeekDone	Acc to Dev	106900, 30	Tuner seek has finished with tuned frequency of 106.9 MHz and RSSI level of 30.
GetHDProgramServiceCount	Dev to Acc	No data	Retrieve number of HD Program Services available.

Command	Direction	Data	Comments
RetHDProgramService-Count	Acc to Dev	3,1	3 HD Program services and the analog programming are available.
SetHDProgramService	Dev to Acc	1	Start decode and playback of HD program service #1.
ACK	Acc to Dev	0x00	Acknowledge SetHDProgramService
Collect info on HD programs, as shown in "HD Radio Service Management" (page 329). Info collection could have been performed before tuning by using SetHDDataNotifyMask; see "Initializing HD Radio" (page 326).			
Set notification:			
SetStatusNotifyMask	Dev to Acc	0x18	Set status notification for HD signal present, HD digital audio present.
ACK	Acc to Dev	0x00	Acknowledge SetStatusNotifyMask
Retrieve HD signal and audio status:			
StatusChangeNotify	Acc to Dev	0x18	Status change notification: HD signal and HD audio present.
GetHDProgramService-Count	Dev to Acc	No data	Retrieve number of HD Program Services available.
RetHDProgramService-Count	Acc to Dev	2,1	2 HD Program services and the analog programming are available.
SetHDProgramService	Dev to Acc	1	Start decode/playback of HD program service #1.
ACK	Acc to Dev	0x00	Acknowledge SetHDProgramService
Respond to loss of HD digital audio:			
StatusChangeNotify	Acc to Dev	0x08	Status change notification: HD Signal present, HD digital audio and data not present.
GetHDProgramService-Count	Dev to Acc	No data	Retrieve number of HD Program Services available.
RetHDProgramService-Count	Acc to Dev	0,1	Zero HD Program services and the analog programming are available.
SetHDProgramService	Dev to Acc	0	Start playback of analog programming.
ACK	Acc to Dev	0x00	Acknowledge SetHDProgramService

HD Radio Service Management

This example retrieves PSD and other data for HD radio, using RF Tuner lingo commands.

Table 3-210 Example of getting PSD and name data

Command	Direction	Data	Comments
SetHDDDataNotifyMask	Dev to Acc	0x00000009	Set HD data ready notification for Station Short Name and PSD Data.
ACK	Acc to Dev	0x00	Acknowledge SetHDDDataNotifyMask
HDDDataReadyNotify	Acc to Dev	0x03, 1, NNNN	Station Short Name received, ISO 8859-1.
HDDDataReadyNotify	Acc to Dev	0x00, 0, 1, 1, NNNN	PSD ID3 Frame data 1 of 2 received for program 1.
HDDDataReadyNotify	Acc to Dev	0x00, 1, 1, 1, NNNN	PSD ID3 Frame data 2 of 2 received for program 1.
HDDDataReadyNotify	Acc to Dev	0x00, 0, 0, 2, NNNN	PSD ID3 Frame data 1 of 1 received for program 2.
GetHDDDataReadyStatus	Dev to Acc	No Data	Ask accessory to return HD data ready status.
RetHDDDataReadyStatus	Acc to Dev	0x00000009	Station Short Name and PSD data available.
GetHDDData	Dev to Acc	0x03	Ask accessory for Station Short Name.
RetHDDData	Acc to Dev	0x03, 1, NNNN	Station Short Name, ISO 8859-1.
GetHDDData	Dev to Acc	0x00	Ask accessory for PSD Data.
RetHDDData	Acc to Dev	0x00, 0, 1, 1, NNNN	PSD ID3 Frame data 1 of 2 received for program 1.
RetHDDData	Acc to Dev	0x00, 1, 1, 1, NNNN	PSD ID3 Frame data 2 of 2 received for program 1.
GetHDDDataReadyStatus	Dev to Acc	No Data	Ask accessory to return HD data ready status.
RetHDDDataReadyStatus	Acc to Dev	0x00000001	PSD data available.
GetHDDData	Dev to Acc	0x00	Ask accessory for PSD Data.
RetHDDData	Acc to Dev	0x00, 0, 0, 2, NNNN	PSD ID3 Frame data 1 of 1 received for program 2.
GetHDDDataReadyStatus	Dev to Acc	No Data	Ask accessory to return HD data ready status.
RetHDDDataReadyStatus	Acc to Dev	0x00000000	No more data available.

Lingo 0x08: Accessory Equalizer Lingo

Apple devices may be connected to accessory devices such as boom boxes or amplifiers that are capable of frequency response equalization. The Accessory Equalizer lingo, number 0x08, lets the Apple device control the Equalizer Settings of the accessory.

Note: The Accessory Equalizer lingo requires accessory authentication for all communication transports (serial or USB).

The Accessory Equalizer lingo is similar to the Display Remote Equalizer lingo, but the command direction is reversed: the Apple device is the master, initiating commands to which the accessory responds. The Apple device can query the total count of accessory Equalizer Settings and the UTF-8 name for each one; it can then get or set each Equalizer Setting on the accessory.

To support this lingo, the Apple device's application software will add an Apple device menu item for accessory equalizer selection and control.

Equalizer Setting Requirements

To be compatible with the Accessory Equalizer lingo, an accessory must observe these rules:

- The accessory must have an Equalizer Setting with an index of 0x00 and this must be the accessory equalizer's off/none setting.
- The Equalizer Index and Equalizer Setting count fields are 1 byte in size, limiting the setting count to a maximum of 255, including the required index 0x00 setting.
- Devices supporting this lingo must support at least two Equalizer Settings: off/none and at least one other setting. If fewer than two settings are returned by the accessory, the Apple device will not present equalizer menu options to the user.

Accessory Equalizer Lingo Commands

[Table 3-211](#) (page 330) lists the commands included in the Accessory Equalizer lingo.

Table 3-211 Accessory Equalizer lingo command summary

Command	ID	Direction	Data length	Protocol version	Authentication required
ACK	0x00	Acc to Dev	4	1.00	Yes
GetCurrentEQIndex	0x01	Dev to Acc	2	1.00	Yes
RetCurrentEQIndex	0x02	Acc to Dev	3	1.00	Yes
SetCurrentEQIndex	0x03	Dev to Acc	3	1.00	Yes
GetEQSettingCount	0x04	Dev to Acc	2	1.00	Yes

Command	ID	Direction	Data length	Protocol version	Authentication required
RetEQSettingCount	0x05	Acc to Dev	3	1.00	Yes
GetEQIndexName	0x06	Dev to Acc	3	1.00	Yes
RetEQIndexName	0x07	Acc to Dev	0xNN	1.00	Yes
Reserved	0x08–0xFF	N/A	N/A	N/A	N/A

Command History of the Accessory Equalizer Lingo

[Table 3-212](#) (page 331) shows the history of command changes in the accessory equalizer lingo:

Table 3-212 Accessory Equalizer lingo command history

Lingo version	Command changes	Features
1.00	Add: 0x00–0x07	Accessory Equalizer Index, count, and name support

Command 0x00: ACK

Direction: Accessory to Apple device

This command is sent by the accessory in response to a command sent from the Apple device. An ACK response is sent when a bad parameter is received, an unsupported/invalid command is received, or a command completed but did not return any data. See [Table 3-213](#) (page 331).

Table 3-213 ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x00	Command ID: ACK
5	0xNN	Status of command received (see Table 3-17 (page 201))
6	0xNN	ID of the command for which the response is being sent
7	0xNN	Checksum

Command 0x01: GetCurrentEQIndex

Direction: Apple device to Accessory

This command is sent by the Apple device to request the current Equalizer Setting from the accessory. In response, the accessory sends a `RetCurrentEQIndex` command with the current Equalizer Index.

The timeout for this command is 2500 ms (2.5 seconds). If the accessory does not respond within the allotted time, the Apple device will retry up to three times. If the accessory does not respond within the specified time and retry count, the command will fail.

Table 3-214 `GetCurrentEQIndex` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x01	Command ID: <code>GetCurrentEQIndex</code>
5	0xF5	Checksum

Command 0x02: RetCurrentEQIndex

Direction: Accessory to Apple device

This command is sent by the accessory in response to the `GetCurrentEQIndex` command received from the Apple device. The Equalizer Index returned may range from 0 (reserved for the off/none Equalizer Setting) to 254 (the maximum possible Equalizer Index). Devices are not required to support 255 settings; they may support as few as the two minimum required settings.

Table 3-215 `RetCurrentEQIndex` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x02	Command ID: <code>RetCurrentEQIndex</code>
5	0xNN	eqIndex: The current accessory Equalizer Index. See Table 3-216 (page 333).
6	0xNN	Checksum

Table 3-216 Accessory Equalizer Setting indices

Index	Description
0x00	Equalizer off/none
0x01	First Equalizer Setting
0xNN	Last Equalizer Setting (accessory-dependent maximum index)

Command 0x03: SetCurrentEQIndex

Direction: Apple device to Accessory

This command is sent by the Apple device to select an accessory Equalizer Index from the supported range. The valid range is from 0 to one less than the Equalizer Setting count returned by the `RetEQSettingCount` command. See [Table 3-216](#) (page 333).

Note: Accessories may receive duplicate `SetCurrentEQIndex` commands at any time and must handle them correctly.

The timeout for this command is 3000 ms (3.0 seconds). If the accessory does not respond within the allotted time, the Apple device will retry up to three times. If the accessory does not respond within the specified time and retry count, the command will fail.

Table 3-217 SetCurrentEQIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x03	Command ID: SetCurrentEQIndex
5	0xNN	eqIndex: the selected accessory Equalizer Index
6	0xNN	Checksum

Command 0x04: GetEQSettingCount

Direction: Apple device to Accessory

This command is sent by the Apple device to determine how many Equalizer Settings the accessory supports. In response, the accessory sends a `RetEQSettingCount` command with the count of Equalizer Settings supported.

The timeout for this command is 3000 ms (3.0 seconds). If the accessory does not respond within the allotted time, the Apple device will retry up to three times. If the accessory does not respond within the specified time and retry count, the command will fail.

Table 3-218 GetEQSettingCount packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x04	Command ID: GetEQSettingCount
5	0xF2	Checksum

Command 0x05: RetEQSettingCount

Direction: Accessory to Apple device

This command is sent by the accessory in response to a GetEQSettingCount command from the Apple device. To support the Accessory Equalizer lingo, an accessory must report a minimum count of 0x02 (equalizer off/none plus one other setting) and may support a maximum count of 0xFF (equalizer off/none plus 254 other settings).

Table 3-219 RetEQSettingCount packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x05	Command ID: RetEQSettingCount
5	0xNN	eqCount: the count of accessory equalizer indices. See Table 3-220 (page 334).
6	0xNN	Checksum

Table 3-220 RetEQSettingCount parameter values

Range	Comment
0x00–0x01	Invalid equalizer count (minimum count is 0x02)

Range	Comment
0x02–0xFF	Valid equalizer count range

Command 0x06: GetEQIndexName

Direction: Apple device to Accessory

This command is sent by the Apple device to obtain the name string associated with a specified Equalizer Index. In response, the accessory sends a `RetEQIndexName` command with the same Equalizer Index and the associated Equalizer Setting name string.

The timeout for this command is 3000 ms (3.0 seconds). If the accessory does not respond within the allotted time, the Apple device will retry up to three times. If the accessory does not respond within the specified time and retry count, the command will fail.

Table 3-221 `GetEQIndexName` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x06	Command ID: <code>GetEQIndexName</code>
5	0xNN	eqIndex: the selected accessory Equalizer Index
6	0xNN	Checksum

Command 0x07: RetEQIndexName

Direction: Accessory to Apple device

This command is sent by the accessory in response to a `GetEQIndexName` command received from the Apple device. The `eqIndex` byte is the same index that was sent by the `GetEQIndexName` command. The `eqName` string is a null-terminated UTF-8 character array. The array length must not exceed 32 characters plus a null terminator character. This length limit minimizes truncation of the Equalizer Setting name when it is displayed in the Apple device's accessory Equalizer Settings menu.

Table 3-222 `RetEQIndexName` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x07	Command ID: RetEQIndexName
5	0xNN	eqIndex: the selected accessory Equalizer Index
6-NN	0xNN...	The accessory equalizer name, as a null-terminated UTF-8 character array
(last byte)	0xNN	Checksum

Lingo 0x09: Sports Lingo

The Sports lingo enables gym cardio equipment to communicate with an Apple device to track a user's workout while the user continues to use the Apple device for other purposes. The Sports lingo version of the Apple device must be 1.01 or later to support these accessories. For information about determining the lingo version, see [“Determining Apple Device Support for Cardio Equipment”](#) (page 569).

All Sports lingo commands require authentication level 2.0.

Sports Lingo commands

[Table 3-223](#) (page 336) summarizes the Sports commands (lingo 0x09).

Table 3-223 Sports lingo command summary

Command	ID	Direction	Data length	Protocol version
DeviceACK	0x00	Acc to Dev	0x02	1.01
GetDeviceVersion	0x01	Dev to Acc	0x00	1.01
RetDeviceVersion	0x02	Acc to Dev	0x02	1.01
GetDeviceCaps	0x03	Dev to Acc	0x00	1.01
RetDeviceCaps	0x04	Acc to Dev	0x03	1.01
Reserved	0x05-0x7F	N/A		
iPodACK	0x80	Dev to Acc	0x02	1.01
Reserved	0x81-0x82	N/A		
Get iPodCaps	0x83	Acc to Dev	0x00	1.01

Command	ID	Direction	Data length	Protocol version
RetiPodCaps	0x84	Dev to Acc	0x03	1.01
GetUserIndex	0x85	Acc to Dev	0x00	1.01
RetUserIndex	0x86	Dev to Acc	0x01	1.01
Reserved	0x87	N/A		
GetUserData	0x88	Acc to Dev	0x01	1.01
RetUserData	0x89	Dev to Acc	0xNN	1.01
SetUserData	0x8A	Acc to Dev	0xNN	1.01
Reserved	0x8B-0xFF	N/A		

Command History of the Sports Lingo

Table 3-224 (page 337) shows the history of changes to the Sports lingo.

Table 3-224 Sports lingo command history

Lingo version	Command changes	Features
1.01	Add: 0x00–0x04, 0x80, 0x83–0x86, 0x88–0x8A	Cardio equipment accessory support

Command 0x00: DeviceACK

Direction: Accessory to Apple device

This command is sent by the accessory in response to a command sent from the Apple device. An ACK response must be sent for any of the following conditions:

- A bad parameter is received
- An unsupported/invalid command is received
- A command that does not return any data has completed

Table 3-225 DeviceACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x04	Length of Packet

Byte number	Value	Comment
3	0x09	Lingo ID: Sports
4	0x00	Command ID: DeviceACK
5	0xNN	ackStatus of command; see Table 3-226 (page 338) for details
6	0xNN	ID of command being acknowledged
7	0xNN	Checksum

Table 3-226 (page 338) presents the possible values for the ackStatus byte.

Table 3-226 ackStatus details

ackStatus	Description
0x00	Success
0x01	N/A (reserved)
0x02	Command failed
0x03	Out of resources
0x04	Bad parameter
0x05	N/A (reserved)
0x06	N/A (reserved)
0x07	Not authenticated
0x08-0xFF	Reserved

Command 0x01: GetDeviceVersion

Direction: Apple device to Accessory

This command is sent by Apple device to obtain the sports accessory lingo protocol version. In response, the accessory will send the RetDeviceVersion command with its major and minor protocol version numbers. Devices may query the Apple device's Sports Lingo protocol version using the General Lingo RequestLingoProtocolVersion command.

Table 3-227 GetDeviceVersion packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)

Byte number	Value	Comment
2	0x02	Length of Packet
3	0x09	Lingo ID: Sports
4	0x01	Command ID: GetDeviceVersion
5	0xF4	Checksum

Command 0x02: RetDeviceVersion

Direction: Accessory to Apple device

This command must be returned by the accessory in response to each `GetDeviceVersion` command received from the Apple device. The accessory returns the maximum lingo version that it supports. The first byte is the major version number (tens/ones digits left of the decimal point) and the second byte is the minor version number (tenths/hundreths digits right of the decimal point). The Apple device will support all accessory lingo versions up to and including its own current lingo version.

Table 3-228 RetDeviceVersion packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x04	Length of Packet
3	0x09	Lingo ID: Sports
4	0x02	Command ID: RetDeviceVersion
5	0xNN	Major Sports lingo protocol version supported by the accessory (currently 0x01)
6	0xNN	Minor Sports lingo protocol version supported by the accessory (currently 0x01)
7	0xNN	Checksum

Command 0x03: GetDeviceCaps

Direction: Apple device to Accessory

This command is sent by the Apple device to get the sports accessory capabilities and determine the features present on the accessory. In response, the accessory will send the `GetDeviceCaps` command with the payload indicating the capabilities supported.

Table 3-229 GetDeviceCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x02	Length of Packet
3	0x09	Lingo ID: Sports
4	0x03	Command ID: GetDeviceCaps
5	0xF2	Checksum

Command 0x04: RetDeviceCaps

Direction: Accessory to Apple device

This command must be sent by the accessory in response to each GetDeviceCaps command sent by the Apple device. The sports accessory returns the payload indicating which of the capabilities it supports. When a capability bit is set, it means that the associated command is supported by the accessory. Conversely, if a capability bit is clear, it means that the associated commands are not supported by the accessory.

Table 3-230 RetDeviceCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x05	Length of Packet
3	0x09	Lingo ID: Sports
4	0x04	Command ID: GetDeviceCaps
5	0xNN	capsMask (bits15:8)
6	0x00	capsMask (bits7:0)
7	0x00	Reserved (set equal to 0x00)
8	0xNN	Checksum

[Table 3-231](#) (page 341) presents the possible values for the capsMask bits.

Table 3-231 RetDeviceCaps capsMask details

capsMask	Description
bits 0-8	Reserved (set to 0)
bit 9	Accessory supports (1) or does not support (0) Sports lingo commands 0x80 through 0x8A
bits 15:10	Reserved (set to 0)

Command 0x80: iPodACK

Direction: Apple device to Accessory

This command is sent by the Apple device in response to a command sent from the accessory. An ACK response is sent when a bad parameter is received, an unsupported/invalid command is received, or a command that does not return any data has completed.

Table 3-232 iPodACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x04	Length of Packet
3	0x09	Lingo ID: Sports
4	0x80	Command ID: iPodACK
5	0xNN	ackStatus of Command
6	0xNN	ID of Command being acknowledged
7	0xNN	Checksum

[Table 3-233](#) (page 341) presents the possible values for the iPodACK byte.

Table 3-233 iPodACK ackStatus details

ackStatus	Description
0x00	Success
0x01	N/A (reserved)
0x02	Command failed
0x03	Out of resources
0x04	Bad parameter

ackStatus	Description
0x05-0xFF	N/A (reserved)

Command 0x83: GetiPodCaps

Direction: Accessory to Apple device

This command may be sent by the accessory to get the Apple device capabilities and determine if required features are present. In response, the Apple device will send the RetiPodCaps command with the payload indicating the capabilities supported.

Note: The GetiPodCaps command must be sent before the accessory may use any commands that rely upon Apple device support for gym cardio equipment features, such as GetUserIndex.

Table 3-234 GetiPodCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x02	Length of Packet
3	0x09	Lingo ID: Sports
4	0x83	Command ID: GetiPodCaps
5	0x72	Checksum

Command 0x84: RetiPodCaps

Direction: Apple device to Accessory

This command is sent by the Apple device in response to the GetiPodCaps command sent by the accessory. The Apple device returns the payload indicating which of the capabilities it supports. The user count is valid only if the Apple device user data capability bit is set.

Table 3-235 RetiPodCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x05	Length of Packet

Byte number	Value	Comment
3	0x09	Lingo ID: Sports
4	0x84	Command ID: RetiPodCaps
5	0x00	capsMask (bits15:8)
6	0xNN	capsMask (bits7:0)
7	0xNN	userCount number of user data profiles on the Apple device
8	0xNN	Checksum

Table 3-236 (page 343) presents the possible values for the capsMask bits.

Table 3-236 RetiPodCaps capsMask details

capsMask	Description
bit 0	Cardio equipment is supported (1) or is not supported (0).
bit 1	User data is supported (1) or is not supported (0). The userCount value is valid if and only if this capability bit is set.
bits 15:2	Reserved (set to 0)

Command 0x85: GetUserIndex

Direction: Accessory to Apple device

This command may be sent by the accessory to obtain the current user index selection from the Apple device. In response, the Apple device will return the RetUserIndex command with the current user index.

Note: This command must not be sent unless the accessory has already sent a GetiPodCaps command and received a RetiPodCaps command indicating user data support capability. If the Apple device does not support user data, it will return an iPodACK with a bad parameter status.

Table 3-237 GetUserIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x02	Length of Packet
3	0x09	Lingo ID: Sports
4	0x85	Command ID: GetUserIndex

Byte number	Value	Comment
5	0x70	Checksum

Command 0x86: RetUserIndex

Direction: Apple device to Accessory

This command is sent by the Apple device in response to the `GetUserIndex` command received from the accessory. The current user index is returned (the first user has index 0). The total user count present on the Apple device is obtained using the `GetPodCaps` command.

Table 3-238 RetUserIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x03	Length of Packet
3	0x09	Lingo ID: Sports
4	0x86	Command ID: RetUserIndex
5	0xNN	userIndex (0 to userCount-1) (see Note below)
6	0xNN	Checksum

Note: The `userIndex` value 0 is reserved by the Apple device for the default/unknown user profile. An Apple device that does not support multiple user profiles but does support user data will offer only `userIndex` 0.

Command 0x88: GetUserData

Direction: Accessory to Apple device

This command may be sent by the accessory to obtain the current user's data. In response, the Apple device will return the `RetUserData` command with the specified `userDataType`. If the Apple device does not support this command or the `userDataType` is not valid, the Apple device will return an `iPodACK` with a bad parameter status.

Note: This command must not be sent unless the accessory has already sent a `GetPodCaps` command and received a `RetPodCaps` command indicating user data support capability. If the Apple device does not support user data, it will return an `iPodACK` with a bad parameter status.

Table 3-239 `GetUserData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x03	Length of Packet
3	0x09	Lingo ID: Sports
4	0x88	Command ID: <code>GetUserData</code>
5	0xNN	<code>userDataType</code>
6	0xNN	Checksum

[Table 3-240](#) (page 345) presents the possible values for the `userDataType` byte.

Table 3-240 `GetUserData` `userDataType` details

<code>userDataType</code>	Description
0x00	Preferred Unit System
0x01	Name
0x02	Gender
0x03	Weight
0x04	Age
0x05	Workout Recording Preference
0x06-0xFF	Reserved

Command 0x89: `RetUserData`

Direction: Apple device to Accessory

This command is sent by the Apple device in response to the `GetUserData` command received from the accessory. The Apple device returns the requested user data for the specified data type.

Table 3-241 RetUserData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0xNN	Length of Packet
3	0x09	Lingo ID: Sports
4	0x89	Command ID: RetUserData
5	0xNN	userDataType
6...N	0xNN	userData (variable length)
(last byte)	0xNN	Checksum

Table 3-242 (page 346) presents the possible values for the userDataType byte and userData field values.

Table 3-242 RetUserData userDataType details

userDataType	userData
0x00	Preferred unit system (1 byte): 0x00 = No information 0x01 = Imperial units (lbs/miles) 0x02 = Metric units (kg/km) 0x03-0xFF = Reserved
0x01	Name (variable length, 128 bytes max, null-terminated UTF-8 string)
0x02	Gender (1 byte): 0x00 = No information 0x01 = Female 0x02 = Male 0x03-0xFF = Reserved
0x03	Weight in tenths of a kg (2 bytes, MSB first): 0x0000 = No information 0x0001 = 0.1 kg 0x0200 = 51.2 kg 0x1388 = 500.0 kg (maximum weight limit) 0x1389-0xFFFF = Reserved

userDataType	userData
0x04	Age in years (1 byte): 0x00 = No information 0x20 = 32 years old 0xC8 = 200 years old (maximum age limit) 0xC9-0xFF = Reserved
0x05	Workout recording preference (1 byte): 0x00 = No information 0x01 = Never record workout data 0x02 = Ask if workout should be recorded 0x03 = Always record workout data 0x04-0xFF = Reserved
0x06-0xFF	Reserved

Command 0x8A: SetUserData

This command may be sent by the accessory to set the current user's data. In response, the Apple device will send the `iPodACK` command with the status of the operation. If the Apple device does not support this command or the `userDataType` or `userData` is not valid, an `iPodACK` with a bad parameter status is returned.

Note: This command must not be sent unless the accessory has already sent a `GetiPodCaps` command and received a `RetiPodCaps` command indicating user data support capability. If the Apple device does not support user data, it will return an `iPodACK` with a bad parameter status.

Table 3-243 SetUserData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0xNN	Length of Packet
3	0x09	Lingo ID: Sports
4	0x8A	Command ID: SetUserData
5	0xNN	userDataType
6...N	0xNN	userData (variable length)
(last byte)	0xNN	Checksum

Table 3-244 (page 348) presents the possible values for the `userDataType` byte and `userData` field values.

Table 3-244 SetUserData `userDataType` details

<code>userDataType</code>	<code>userData</code>
0x00	Preferred unit system (1 byte): 0x00 = No information 0x01 = Imperial units (lbs/miles) 0x02 = Metric units (kg/km) 0x03-0xFF = Reserved
0x01	Setting not allowed
0x02	Gender (1 byte): 0x00 = No information 0x01 = Female 0x02 = Male 0x03-0xFF = Reserved
0x03	Weight in tenths of a kg (2 bytes, MSB first): 0x0000 = No information 0x0001 = 0.1 kg 0x0200 = 51.2 kg 0x1388 = 500.0 kg (maximum weight limit) 0x1389-0xFFFF = Reserved
0x04	Age in years (1 byte): 0x00 = No information 0x20 = 32 years old 0xC8 = 200 years old (maximum age limit) 0xC9-0xFF = Reserved
0x05	Setting not allowed
0x06-0xFF	Reserved

Lingo 0x0A: Digital Audio Lingo

Digital Audio lingo commands are used to configure USB Device Mode audio on Apple devices which support that feature (see ["General Apple Device Features"](#) (page 44)). The Apple device uses these lingo commands to retrieve a list of supported sample rates from the accessory and to inform the accessory of the Apple device's current sample rate, Sound Check value, and track volume adjustment value (which is set using

iTunes). After these interchanges, the Apple device performs audio sample rate conversion internally and transfers digital audio to the accessory at one of the accessory's supported audio data sample rates. For a sample command sequence that declares this lingo, see [Table 3-312](#) (page 406).

Note: The Apple devices that contain version 1.00 of the Digital Audio lingo do not correctly support digital audio. An accessory should check the attached Apple device's version of the Digital Audio lingo and use digital audio only if the version number is greater than 1.00. Version 1.01 of Digital Audio may be used only while the Apple device is in Extended Interface mode (Lingo 0x04); later versions do not have this restriction. See [Table 1-2](#) (page 42) and [Table 1-3](#) (page 43) for lists of affected Apple device models.

Accessory Authentication

Every accessory that supports the Digital Audio lingo must authenticate itself with a connected Apple device as soon as the Apple device recognizes the accessory; deferred authentication is not permitted. This means that the accessory's IDPS process must specify authentication immediately after identification in its `IdentifyToken` (see [Table 2-78](#) (page 133)). With Apple devices that do not support IDPS, the accessory must send an `IdentifyDeviceLingoes` command with the authentication control bits in its `options` field set to 10_b (Authenticate immediately after identification). See [Table 2-34](#) (page 98) for details.

Note: The General lingo `Identify` command cannot be used to identify an accessory for any purpose if the accessory supports the Digital Audio lingo.

When the accessory identifies itself as supporting the Digital Audio lingo, authentication can happen in the background and the Apple device can proceed to transfer digital audio as if authentication were successful.

Digital Audio Lingo Commands

The Digital Audio lingo ID is 0x0A. All its commands are transferred in small packet format and all require authentication. They are listed in [Table 3-245](#) (page 349).

Table 3-245 Digital Audio lingo command summary

ID	Command	Data length	Protocol version	Authentication required
0x00	AccAck	0x04	1.00	Yes
0x01	iPodAck	0x04	1.00	Yes
0x02	GetAccSampleRateCaps	0x02	1.00	Yes
0x03	RetAccSampleRateCaps	NN	1.00	Yes
0x04	TrackNewAudioAttributes	0x0E	1.00	Yes
0x05	SetVideoDelay	0x06	1.03	Yes
0x06–0xFF	Reserved	N/A	N/A	N/A

Command History of the Digital Audio Lingo

Table 3-246 (page 350) shows the history of command changes in the Digital Audio lingo:

Table 3-246 Digital Audio lingo command history

Lingo version	Command changes	Features
1.00	Add: 0x00–0x04	Digital audio sample rate and track information support
1.01	None	The <code>TrackNewAudioAttributes</code> command is resent until it is acknowledged by the USB host with an <code>AccAck</code> command. Also corrected a bug where <code>TrackNewAudioAttributes</code> was not being sent before every track; for this reason, accessories should not use Digital Audio with Apple devices that support only Digital Audio lingo version 1.00.
1.02	None	The Digital Audio lingo no longer requires the Apple device to be in Extended Interface mode.
1.03	Add 0x05	Added <code>SetVideoDelay</code> to allow digital audio to synchronize with video playback.

USB Device Mode Audio

When an Apple device is in USB Device mode, it can stream digital audio to an accessory as a USB Audio 1.0 peripheral. The audio stream contains PCM data and is synchronized using USB 1 ms start-of-frame (SOF) packets. The Digital Audio lingo must be used to drive this process. For transport details, see “Using an Apple Device as a USB Device” in *MFi Accessory Hardware Specification*.

Note: The Apple device USB isochronous audio data endpoint descriptor `bmAttributes` field erroneously returns the Synchronization Type field (D3:2) as b00 (no synchronization) instead of the correct value, b11 (synchronous). The Apple device supports synchronous data transfers, so USB host devices must override these attribute bits. The erroneous b00 value is retained for backwards compatibility with older accessories.

Note: Any accessory that outputs digital audio obtained from an Apple device must implement copy protection in its output stream; for example, by setting Serial Copy Management System (SCMS) bits to 10.

When receiving digital audio over a USB audio streaming interface, the accessory may buffer data to achieve consistent audio playback. Since buffering introduces latency, it is suggested that this latency be kept below 200 ms to ensure timely response to user input. Apple may enforce this suggestion in the future.

Digital Audio and Extended Interface Mode

Digital Audio lingo version 1.01 requires the Apple device to be in Extended Interface mode before USB audio can be transferred. With versions 1.02 and later, USB audio can be transferred in any mode.

Audio/Video Synchronization

Digital Audio lingo versions 1.03 and above allow synchronization between Apple device video and the sound for that video that is being transmitted as digital audio. The video may either appear on the Apple device's display or be transmitted through the analog video output.

Accessories that support both video output and digital audio streaming from an Apple device must maintain synchronized digital audio/video playback whether or not the Apple device supports the Digital Audio lingo `SetVideoDelay` command. To do this, the accessory must first determine the Apple device's audio routing support, after which the accessory can select the best way to maintain digital audio/video synchronization. To achieve the best available Apple device audio routing, the accessory should do the following:

1. Send a General lingo `StartIDPS` command to begin the accessory identification process.
2. Send a General lingo `GetiPodOptionsForLingo` command for the digital audio lingo (0x0A). If the Apple device returns a `RetiPodOptionsForLingo` command with bit 00 set to 1, continue to Step 3; otherwise go to Step 4.
3. If the Digital Audio lingo `SetVideoDelay` support bit (bit 00) is set to 1, the Apple device supports using the `SetVideoDelay` command to synchronize the video and digital audio streams. The accessory must use that command to maintain synchronization between the video and digital audio streams by sending a fixed time (in milliseconds) that the Apple device will add as a delay to its own video stream. The accessory must also resend the command whenever its audio latency changes, for example at sample rate changes. The delay should be equal to the audio latency of the accessory; it corresponds to the difference between the time that digital audio data appears on the digital audio transport (USB) and the time the corresponding analog audio is sent to the speakers. There is no need to switch to an analog line out when video media is playing. After sending `SetVideoDelay`, go to Step 7.
4. If the Apple device does not support the Digital Audio lingo `SetVideoDelay` command, the accessory must switch to its analog line out when video media is playing. The accessory must send a General lingo `GetiPodOptionsForLingo` command for the General lingo (0x00). If the Apple device returns a `RetiPodOptionsForLingo` command with bit 26 set to 1, continue to Step 5; otherwise go to Step 6.
5. If the General lingo analog/USB audio routing support bit (Bit 26) is set to 1, the Apple device supports using the `SetiPodPreferences` line out preference (class ID 0x03) to switch between analog line out and USB digital audio routings. The accessory does not need to identify or authenticate itself again. The accessory must include the digital audio lingo (0x0A) in its FID `IdentifyToken` and must set the analog/USB audio routing bit (bit 21) in its FID `AccCapsToken`. Go to Step 7.
6. The Apple device does not support switching its audio routing using the `SetiPodPreferences` command. To switch between USB digital audio and analog line out audio routings, the accessory must identify and authenticate itself without declaring support for the Digital Audio lingo. Continue to Step 7.
7. Resume the IDPS process, including any FID token settings required in Step 5.

Line Level Output

Line level output and USB Device Mode audio are mutually exclusive. All Apple device audio is sent to the LINE-OUT pins of the 30-pin connector until digital audio transport is enabled, at which point the Apple device's audio output is redirected to the USB audio streaming interface. When USB Device Mode audio is disabled, Apple device audio is again sent to the LINE-OUT pins (see [Table 3-1](#) (page 183)).

When disabling USB Device Mode audio, the Apple device does not automatically enter the playback Pause state, but instead continues in its current Play or Pause state. The only difference is that the output mode changes to LINE-OUT. If appropriate, the accessory should set the Apple device to Pause when disabling digital audio.

Enabling and Disabling USB Device Mode Audio

USB Device Mode audio is enabled and LINE-OUT audio is disabled when:

- Digital Audio lingo version 1.01: The accessory selects a nonzero-bandwidth alternate USB audio streaming interface and enters Extended Interface mode.
- Digital Audio lingo version 1.02 and later: The accessory replies to a `GetAccSampleRateCaps` command with a list of valid sample rates.

USB Device Mode audio is disabled and LINE-OUT audio is enabled when the accessory either reidentifies itself without supporting the Digital Audio lingo or disconnects from the USB Audio Configuration (for example, the user unplugs the USB cable).

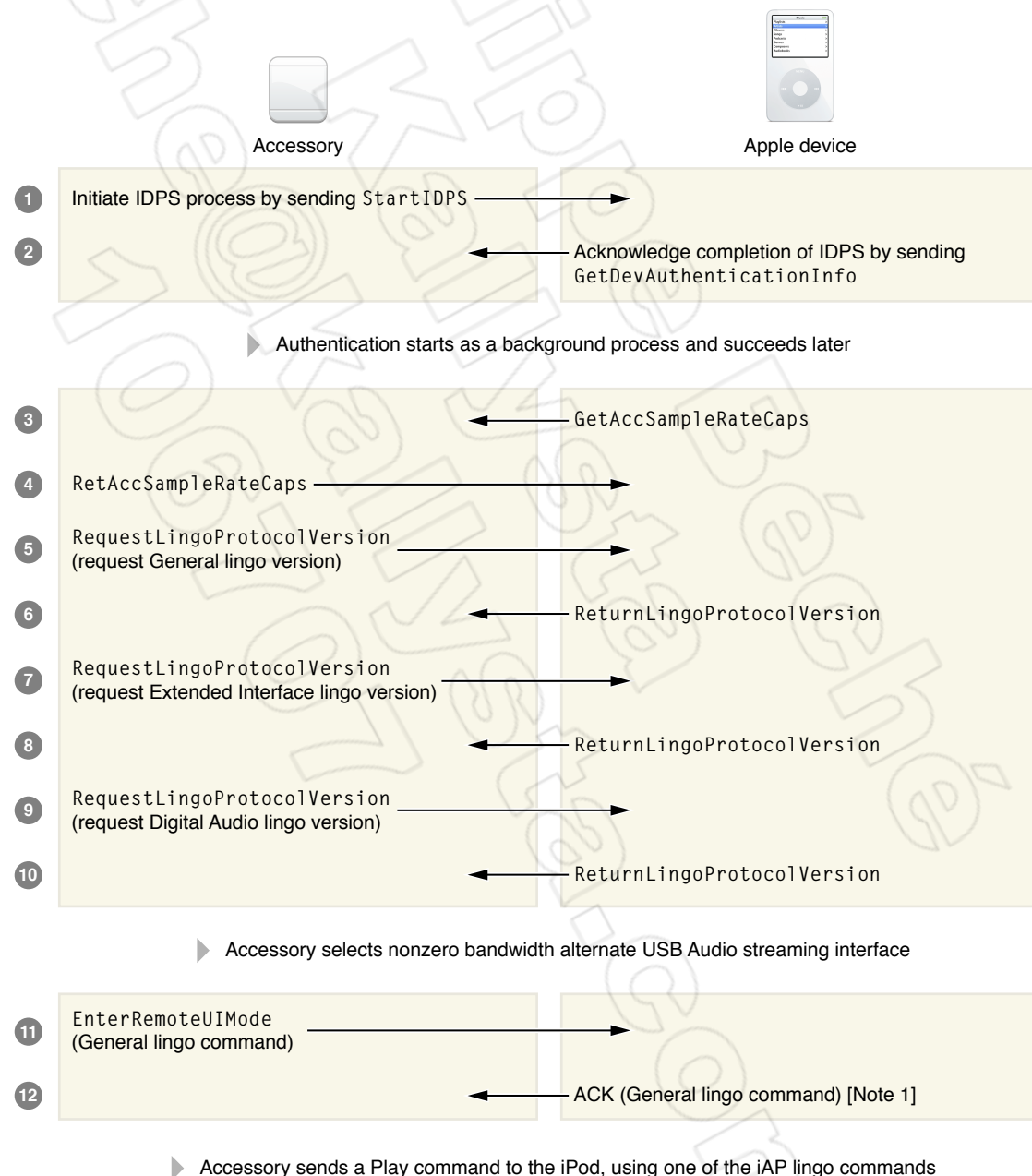
To enable USB Device Mode audio, an Apple device and its attached accessory must perform the following steps, in the order listed, after the user has connected the Apple device to the accessory:

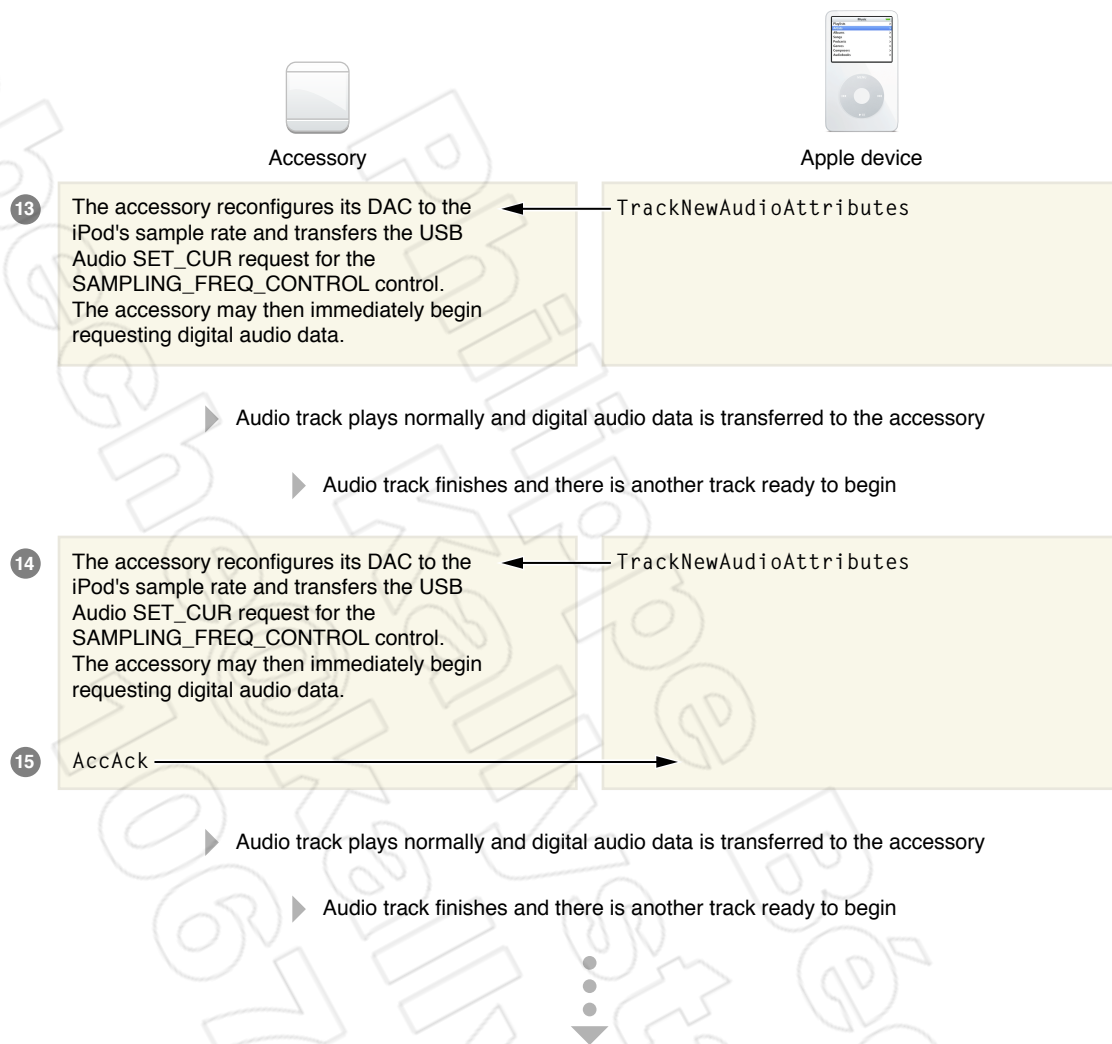
1. The Apple device's USB enumeration lists two configurations: 1 = Mass Storage, 2 = USB Audio and HID device.
2. The accessory sends the USB standard request `Set_Configuration` to select configuration 2.
3. The accessory completes the IDPS identification process, declaring the General lingo and the Digital Audio lingo, plus the Extended Interface lingo if the Apple device does not support Digital Audio lingo version 1.02. With Apple devices that do not support IDPS, the `IdentifyDeviceLingoes` command may be used. The Apple device authenticates the accessory for the requested lingoes.
4. The accessory selects a nonzero-bandwidth alternate USB audio streaming interface on the Apple device.
5. The accessory tells the Apple device to enter Extended Interface Mode using the General lingo `EnterRemoteUIMode` command (not required for Digital Audio lingo version 1.02). If an audio track is playing, playback will pause. If a video track is playing, playback will stop.
6. The accessory places the Apple device in the Play state.
7. The Apple device sends a `TrackNewAudioAttributes` command to the accessory.
8. The accessory acknowledges the `TrackNewAudioAttributes` command using the `AccAck` command.
9. The accessory reconfigures its DAC to the Apple device's sample rate and transfers the USB Audio `SET_CUR` request for the `SAMPLING_FREQ_CONTROL` control, passing a sample rate equal to the value sent by the `TrackNewAudioAttributes` command.
10. The accessory requests digital audio packets from the isochronous USB pipe.
11. The audio track plays normally and digital audio data is transferred to the accessory.
12. The audio track finishes and there is another track ready to begin.
13. Steps 7 through 12 may be repeated indefinitely.

Note: Step 2 is slightly different if a 191 k Ω resistor is used as the identify resistor (see “Accessory Detect and Identify” in *MFi Accessory Hardware Specification*). In this case, the Apple device will present the USB audio and HID device configuration as the first configuration, and the accessory will be required to select configuration 1 through the `Set_Configuration` USB standard request.

An example of this process is diagrammed in [Figure 3-1](#) (page 353), where the transactions shown assume that USB enumeration and accessory authentication are successful. An example of the Digital Audio identification process is shown in [Table 3-312](#) (page 406).

Figure 3-1 Typical digital audio transactions between an Apple device and an accessory





Note 1: The ACK reply to an `EnterRemoteUIMode` command may return a command pending acknowledgment along with a waiting time.

Note: With Digital Audio lingo version 1.01, switching from the zero bandwidth alternate USB audio streaming interface to the nonzero bandwidth interface without first exiting and entering Extended Interface Mode can prevent a `TrackNewAudioAttributes` command from being sent from the Apple device when a track change occurs. The accessory must not select the zero-bandwidth alternate interface on the Apple device, even when changing sample rates, unless it is also exiting Extended Interface Mode.

With Digital Audio lingo version 1.01, reenabling digital audio after an accessory has disabled it requires the following actions:

1. Enable the nonzero-bandwidth alternate USB audio streaming interface on the Apple device.
2. Enter Extended Interface Mode using the General lingo `EnterRemoteUIMode`. This will pause Apple device playback.
3. Place the Apple device in the Play state.

4. Receive a `TrackNewAudioAttributes` command from the Apple device.
5. Acknowledge the `TrackNewAudioAttributes` command using the `AccAck` command.
6. Configure the accessory's playback DAC to the Apple device's sample rate.
7. Send a USB audio `SET_CUR` request for the `SAMPLING_FREQ_CONTROL` control, passing a sample rate equal to the value sent by the `TrackNewAudioAttributes` command.
8. Request digital audio packets from the isochronous USB pipe.

If the accessory requests digital audio data from the isochronous USB pipe before digital audio is enabled or before the correct digital sample rate has been negotiated, the Apple device will return isochronous packets filled with zeros. The Apple device will also return packets filled with zeros if authentication fails.

USB Device Mode Audio Errors on Older Apple Devices

On first-generation nano and 5G iPods, USB Device Mode audio data from the Apple device will occasionally contain 16-bit cyclic redundancy check (CRC16) errors. If iAP commands are sent from the Apple device to a USB host while USB Device Mode audio is enabled, there is a possibility that the USB audio `DATA0` packet immediately preceding the iAP command will contain a CRC16 error. This error occurs infrequently; however, enabling Apple device notifications over USB (such as the Lingo `0x04 PlayStatusChangeNotification` command, which is sent every 500 ms when enabled) greatly increases the possibility that a CRC16 error will occur.

To resolve this problem, the USB host must be able to recover immediately from a CRC16 error in the payload of the isochronous USB Device Mode audio data. [Figure 3-2](#) (page 356) illustrates a typical recovery sequence.

Figure 3-2 A USB host recovers from a CRC16 error

Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time	Time Stamp	
34414	S	IN	1	1	176	1 packets ranging from 176 bytes to 176 bytes	999,900 µs	02109.6563 5780	
Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time	Time Stamp	
34415	S	IN	1	1	176	1 packets ranging from 176 bytes to 176 bytes	999,900 µs	02109.6571 5774	
Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time	Time Stamp	
34416	S	IN	1	1	176	1 packets ranging from 176 bytes to 176 bytes	999,900 µs	02109.6579 5768	
Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time Stamp		
34417	S	IN	1	1	0	1 packets ranging from 0 bytes to 0 bytes	02109.6587 5762		
Transaction	H	IN	ADDR	ENDP	T	Data	Error	Time Stamp	
309165	S		0x96	1	1	0 0 bytes	CRC16 Error	02109.6587 5762	
Packet	Dir	H	IN	ADDR	ENDP	CRC16	Pkt Len	Idle	Time Stamp
893211	-->	S	0x96	1	1	0x1A	8	366,660 ns	02109.6587 5762
Packet	Dir	H	DATA0	Data	CRC16	Pkt Len	Time	Time Stamp	
893212	-->	S	0xC3	95 bytes	0x7DA6	106	16,200 µs	02109.6587 5792	
Transfer	H	Interrupt	ADDR	ENDP	Bytes Transferred	Time Stamp			
34418	S	IN	1	2	14	02109.6587 5764			
Transaction	H	IN	ADDR	ENDP	T	Data	ACK	Time	
309166	S		0x96	1	2	1 03 00 55 08 04 00 27 04 00 03 A1 97 8E 00	0x48	983,233 µs	
Time Stamp									
02109.6587 6764									
Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time	Time Stamp	
34419	S	IN	1	1	176	1 packets ranging from 176 bytes to 176 bytes	999,900 µs	02109.6595 5758	
Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time	Time Stamp	
34420	S	IN	1	1	176	1 packets ranging from 176 bytes to 176 bytes	999,900 µs	02109.6603 5752	
Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time	Time Stamp	
34421	S	IN	1	1	176	1 packets ranging from 176 bytes to 176 bytes	999,933 µs	02109.6611 5746	
Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time	Time Stamp	
34422	S	IN	1	1	180	1 packets ranging from 180 bytes to 180 bytes	999,867 µs	02109.6619 5742	
Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time	Time Stamp	
34423	S	IN	1	1	176	1 packets ranging from 176 bytes to 176 bytes	999,917 µs	02109.6627 5734	
Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time	Time Stamp	
34424	S	IN	1	1	176	1 packets ranging from 176 bytes to 176 bytes	1,001 ms	02109.6635 5729	
Transfer	H	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet info	Time	Time Stamp	

PlayStatusChangeNotification (lingo 4)

Command 0x00: AccAck

Direction: Accessory to Apple device

The accessory sends the `AccAck` command to acknowledge the receipt of a command from the Apple device and returns the command status (see [Table 3-248](#) (page 357)). An `AccAck` command should be sent only for commands whose documentation specifies that an `AccAck` command is needed.

Table 3-247 AccAck packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x04	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x00	Command ID: <code>AccAck</code>
5	0xNN	Command status. See Table 3-248 (page 357)

Byte number	Value	Comment
6	0xNN	The ID for the command being acknowledged
7	0xNN	Checksum

Table 3-248 AccAck status values

status	Meaning
0x00	Success (OK)
0x01	Reserved
0x02	ERROR: Command failed
0x03	ERROR: Out of resources
0x04	ERROR: Bad parameter
0x05	ERROR: Unknown ID
0x06	Reserved
0x07	ERROR: Accessory not authenticated
0x08–0xFF	Reserved

Command 0x01: iPodAck

Direction: Apple device to Accessory

The Apple device sends the `iPodAck` command when it receives an invalid or unsupported command or a bad parameter. The command status returns are the same as those used for the `AccAck` command (see [Table 3-248](#) (page 357)).

Table 3-249 iPodAck packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x04	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x01	Command ID: <code>iPodAck</code>
5	0xNN	Command status. See Table 3-248 (page 357)

Byte number	Value	Comment
6	0xNN	The ID for the command being acknowledged
7	0xNN	Checksum

Command 0x02: GetAccSampleRateCaps

Direction: Apple device to Accessory

The Apple device uses this command to request the list of supported sample rates from the accessory. The Apple device sends it after the accessory indicates its support of the Digital Audio lingo during its identification process. The accessory responds to this command using the `RetAccSampleRateCaps` command.

After the Apple device sends a `GetAccSampleRateCaps` command, it waits up to 3 seconds for a `RetAccSampleRateCaps` command from the accessory before timing out. If a timeout occurs, the Apple device resends the command. It will resend the command up to three times if necessary, and if all three attempts fail, the Apple device will then respond to any digital audio request with zeros.

If the Apple device receives a `RetAccSampleRateCaps` command with invalid parameters, it sends the appropriate command status using an `iPodAck` command (see [Table 3-248](#) (page 357)). At this point the accessory should send a `RetAccSampleRateCaps` with correct parameters. The Apple device allows up to three retries if necessary, and if all three attempts fail, the Apple device then responds to any digital audio request with zeros.

Table 3-250 `GetAccSampleRateCaps` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x02	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x02	Command ID: <code>GetAccSampleRateCaps</code>
5	0xF2	Checksum

Command 0x03: RetAccSampleRateCaps

Direction: Accessory to Apple device

An accessory sends the `RetAccSampleRateCaps` command in response to a `GetAccSampleRateCaps` command from an Apple device.

The accessory returns the list of sample rates it supports with this command. The sample rates must be taken from the list of Apple device sample rates shown in [Table 3-252](#) (page 359). Any audio encoded at an unsupported sample rate is resampled internally by the Apple device to conform to a supported sample rate. In general, audio quality will be better when no resampling is performed; therefore accessories should support as many values listed in [Table 3-252](#) (page 359) as possible.

Note: At a minimum, every accessory must support the sample rates 32 KHz, 44.1 KHz, and 48 KHz.

A `RetAccSampleRateCaps` command with sample rates not listed in [Table 3-252](#) (page 359), or missing any of the required sample rates, is invalid. If the Apple device receives such a command, it sends the accessory an `iPodAck` command with a negative acknowledgment as the command status.

Table 3-251 `RetAccSampleRateCaps` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	2+4n	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x03	Command ID: <code>RetAccSampleRateCaps</code>
5 ... (5+4n-1)	0xNNNNNNNN	A list of n sample rates (32-bit big-endian format) supported by the accessory. The sample rates must be taken from Table 3-252 (page 359).
5+4n	0xNN	Checksum

Table 3-252 Digital audio sample rates supported by Apple devices (in Hertz)

Decimal	Hexadecimal	Required/Optional
8000	0x00001F40	Optional
11025	0x00002B11	Optional
12000	0x00002EE0	Optional
16000	0x00003E80	Optional
22050	0x00005622	Optional
24000	0x00005DC0	Optional
32000	0x00007D00	Required
44100	0x0000AC44	Required
48000	0x0000BB80	Required

Command 0x04: TrackNewAudioAttributes

Direction: Apple device to Accessory

The Apple device sends the `TrackNewAudioAttributes` command before the first audio track begins playing. It sends the command again whenever it starts playing a track with different sample rate, sound check, or track volume parameters. In response to this command, accessories should prepare themselves to receive audio data with the new parameters.

The accessory must acknowledge this command, using the `AccAck` command, but the Apple device does not wait for this acknowledgment before allowing digital audio to be transferred to the accessory.

The sample rate sent to the accessory is taken from the list of sample rates returned to the Apple device by the `RetAccSampleRateCaps` command. If the accessory supports the sample rate of the current audio track, then it is sent as the current sample rate. If the accessory does not support the sample rate, the Apple device resamples the audio data to a supported sample rate in real time and sends this new supported sample rate as the current sample rate.

The Sound Check value and track volume adjustment value are the corresponding values set by iTunes, rounded to the nearest integer. The values represent gain (in decibels) and may be positive or negative. Note that if the Sound Check option in the Apple device is disabled, the `TrackNewAudioAttributes` command always sends 0 as the new Sound Check value.

When the Apple device sends a `TrackNewAudioAttributes` command, it waits up to 500 ms for the `AccAck` command before timing out. If a timeout occurs or if the `AccAck` returns an error as the command status, the Apple device sends the command again.

IMPORTANT: The `TrackNewAudioAttributes` command should not be used as a general mechanism to detect track changes. Use appropriate commands from the Display Remote or Extended Interface lingoes instead.

Table 3-253 `TrackNewAudioAttributes` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x0E	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x04	Command ID: <code>TrackNewAudioAttributes</code>
5	0xNN	New sample rate, bits 31:24
6	0xNN	New sample rate, bits 23:16
7	0xNN	New sample rate, bits 15:8
8	0xNN	New sample rate, bits 7:0

Byte number	Value	Comment
9	0xNN	New Sound Check value, bits 31:24
10	0xNN	New Sound Check value, bits 23:16
11	0xNN	New Sound Check value, bits 15:8
12	0xNN	New Sound Check value, bits 7:0
13	0xNN	New track volume adjustment, bits 31:24
14	0xNN	New track volume adjustment, bits 23:16
15	0xNN	New track volume adjustment, bits 15:8
16	0xNN	New track volume adjustment, bits 7:0
17	0xNN	Checksum

Command 0x05: SetVideoDelay

Direction: Accessory to Apple device

The accessory sends this command to inform the Apple device of a new delay (in milliseconds) that must be applied to Apple device video to synchronize it with the audio for the currently playing video. See ["Audio/Video Synchronization"](#) (page 351). The Apple device responds to `SetVideoDelay` with an `"iPodAck"` (page 357) command.

IMPORTANT: Before sending this command, the accessory must verify that `RetiPodOptionsForLingo` reports that bit 00 for Lingo 0x0A is set; see [Table 2-138](#) (page 167).

Table 3-254 SetVideoDelay packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x06	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x05	Command ID: SetVideoDelay
5	0xNN	Video delay in milliseconds, bits 31:24
6	0xNN	Video delay in milliseconds, bits 23:16
7	0xNN	Video delay in milliseconds, bits 15:8

Byte number	Value	Comment
8	0xNN	Video delay in milliseconds, bits 7:0
9	0xNN	Checksum

Lingo 0x0C: Storage Lingo

The iAP Storage lingo, Lingo 0x0C, lets an accessory store files on an attached Apple device as if it were a hard drive. Use of the Storage lingo requires accessory authentication.

Command History of the Storage Lingo

Table 3-255 (page 362) shows the history of command changes in the Storage lingo:

Table 3-255 Storage lingo command history

Lingo version	Command changes	Features
1.01	Add: 0x00-0x02, 0x04, 0x07-08, 0x10-0x12	Support for iTunes Tagging.
1.02	Add: 0x80-0x82. Add options to 0x12	Add support for Sports lingo.

Command Summary

Table 3-256 (page 362) lists the Storage lingo commands.

Table 3-256 Storage lingo commands

CmdID	Name	Direction	Payload:bytes
0x00	iPodACK	Dev to Acc	[ackStatus:1, cmdID:1, handle:1, (transactionID:2)]
0x01	GetiPodCaps	Acc to Dev	[None]
0x02	RetiPodCaps	Dev to Acc	[totalSpace:8, maxFileSize:4, maxWriteSize:2, Reserved:6, majorVersion:1, minorVersion:1]
0x03	Reserved		
0x04	RetiPodFileHandle	Dev to Acc	[handle:1]
0x05-0x06	Reserved		
0x07	WriteiPodFileData	Acc to Dev	[offset:4, handle:1, data:<var>]

CmdID	Name	Direction	Payload:bytes
0x08	CloseiPodFile	Acc to Dev	[handle:1]
0x09-0x0F	Reserved		
0x10	GetiPodFreeSpace	Acc to Dev	[None]
0x11	RetiPodFreeSpace	Dev to Acc	[freeSpace:8]
0x12	OpeniPodFeatureFile	Acc to Dev	[feature: 0xNN]
0x13-0x7F	Reserved		
0x80	DeviceACK	Acc to Dev	[ackStatus:1, cmdID:1, handle:1]
0x81	GetDeviceCaps	Dev to Acc	[feature: 0xNN]
0x82	RetDeviceCaps	Acc to Dev	[feature: 0xNN]
0x83-0xFF	Reserved		

The following is the typical sequence of commands used to store data on an Apple device:

1. Complete the accessory identification and authentication processes, as specified in "[Accessory Signaling and Initialization](#)" (page 66), making sure to identify the accessory as supporting the Storage lingo.
2. Send `GetiPodCaps` and receive `RetiPodCaps`, which returns the maximum write size.
3. Send `OpeniPodFeatureFile` and receive `RetiPodFileHandle`. The returned file handle points to a specific area in the Apple device's file system. Currently the only accessible area is `/iPod_Control/Device/Accessories/Tags/`, used by the Radio Tagging System (see "[iTunes Tagging](#)" (page 535)).
4. Send one or more `WriteiPodFileData` commands with the data to be stored. Receive an `iPodACK` command for each one.
5. Send `CloseiPodFile` when finished writing all the data. Alternately, all files are automatically closed when the accessory detaches.

Note: The accessory should use "[Command 0x10: GetiPodFreeSpace](#)" (page 369) to verify that there is adequate free space available on the Apple device's drive before attempting to open or write data to a file. If there is less than 5 MB available, then calls to `OpeniPodFeatureFile` or `WriteiPodFileData` may fail for lack of memory. The recommended practice is to check the free space before opening or writing data, and to handle any out-of-memory ACK commands by retaining the data in internal storage and displaying an "iPod Full" message on the accessory's display.

Command 0x00: iPodACK

Direction: Apple device to Accessory

The Apple device sends this command to acknowledge the receipt of a Storage lingo command from the accessory.

Table 3-257 Storage iPodACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x00	Command ID: iPodACK
5	0xNN	Command result status. See Table 3-258 (page 364).
6	0xNN	The ID for the command being acknowledged.
7	0xNN	The file handle for the command (0xFF if not applicable.)
8	0xNN	Checksum

Table 3-258 iPodACK responses

Value	Description
0x00	Success
0x01	N/A (reserved)
0x02	Command failed
0x03	Out of resources
0x04	Bad parameter
0x05	Unknown ID
0x06	N/A (reserved)
0x07	Not authenticated
0x08	Bad authentication version
0x09	N/A (reserved)
0x0A	N/A (reserved)
0x0B	N/A (reserved)
0x0C	N/A (reserved)

Value	Description
0x0D	Invalid file handle
0x0E-0xFF	Reserved

Command 0x01: GetiPodCaps

Direction: Accessory to Apple device

The accessory asks the Apple device to return its storage capabilities. The Apple device replies with RetiPodCaps.

Table 3-259 GetiPodCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x01	Command ID: GetiPodCaps
5	0xF1	Checksum

Command 0x02: RetiPodCaps

Direction: Apple device to Accessory

The Apple device tells the accessory about the Apple device's storage capabilities:

- `totalSpace` is the amount of storage on the Apple device in bytes, including space currently in use.
- `maxFileSize` is the largest possible size, in bytes, of any file on the Apple device.
- `maxWriteSize` is the largest amount of data, in bytes, that can be written to the Apple device in a single `WriteiPodFileData` command.
- `majorVersion` and `minorVersion` are the version number of the Storage lingo protocol implemented by the accessory.

Table 3-260 RetiPodCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0x18	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x02	Command ID: RetiPodCaps
5	0xNN	totalSpace (bits 63:56). The total amount of storage space on the Apple device in bytes, including space currently in use.
6	0xNN	totalSpace (bits 55:48)
7	0xNN	totalSpace (bits 47:40)
8	0xNN	totalSpace (bits 39:32)
9	0xNN	totalSpace (bits 31:24)
10	0xNN	totalSpace (bits 23:16)
11	0xNN	totalSpace (bits 15:8)
12	0xNN	totalSpace (bits 7:0)
13	0xNN	maxFileSize (bits 31:24). The size in bytes of the largest possible file on the Apple device.
14	0xNN	maxFileSize (bits 23:16)
15	0xNN	maxFileSize (bits 15:8)
16	0xNN	maxFileSize (bits 7:0)
17	0xNN	maxWriteSize (bits 15:8). The size in bytes of the largest possible amount of data than can be sent to the Apple device with WriteiPodFile.
18	0xNN	maxWriteSize (bits 7:0)
19	0xNN	Reserved.
20	0xNN	Reserved.
21	0xNN	Reserved.
22	0xNN	Reserved.
23	0xNN	Reserved.
24	0xNN	Reserved.
25	0xNN	majorVersion (1 byte). The major version number.
26	0xNN	minorVersion (1 byte). The minor version number.

Byte number	Value	Comment
27	0xNN	Checksum

Command 0x04: RetiPodFileHandle

Direction: Apple device to Accessory

The Apple device returns a unique 8-bit handle to identify the file.

The value of `handle` is a session-specific identifier that represents a file. This is similar to a Unix file descriptor. The handle is valid until either the accessory is detached or the accessory sends a `CloseiPodFile` command with this handle.

Table 3-261 RetiPodFileHandle packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x04	Command ID: RetiPodFileHandle
5	0xNN	handle (1 byte)
6	0xNN	Checksum

Command 0x07: WriteiPodFileData

Direction: Accessory to Apple device

The accessory writes a block of data to a file starting at the offset passed with this command. The file must have been previously opened for writing; failure is reported as a bad parameter (see [Table 3-258](#) (page 364)). If the file was not previously opened for writing, or the handle is invalid (invalid handle error), or the `writeSize` exceeds the Apple device's capabilities (bad parameter error), then the operation will fail. If the caller attempts to write too much data, the state of the file will be undefined; some or none of the data may have been added to the file.

All data must be written sequentially, but write actions may occur across multiple `WriteiPodFileData` commands. The amount of data transferred must also be within the bounds set by the Apple device's capabilities. If any of these criteria are not met, the Apple device will return an `iPodACK` command with a failure message.

When a `WriteiPodFileData` command fails, the state of the file is left unknown. The accessory must close the file and then create a new file to write the data.

`WriteiPodFileData` passes the following parameters:

- `offset` is the offset (in bytes) into the file at which to begin writing.
- `handle` is a unique file identifier.
- `data` is the data to be written to the file.

Table 3-262 `WriteiPodFileData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x07	Command ID: <code>WriteiPodFileData</code>
5	0xNN	offset (bits 31:24). The offset into the file at which to begin writing, in bytes.
6	0xNN	offset (bits 23:16)
7	0xNN	offset (bits 15:8)
8	0xNN	offset (bits 7:0)
9	0xNN	handle (1 byte).
10...N	0xNN	data (variable length). The file data.
(last byte)	0xNN	Checksum

Command 0x08: `CloseiPodFile`

Direction: Accessory to Apple device

This command closes a file and releases its handle. The handle is invalid for further use after this call, and the Apple device may assign the handle later to represent a different file. An `iPodACK` command is sent on success or failure. Parameter `handle` is a unique file identifier.

Table 3-263 `CloseiPodFile` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet

Byte number	Value	Comment
3	0x0C	Lingo ID: Storage lingo
4	0x08	Command ID: CloseiPodFile
5	0xNN	handle (1 byte).
6	0xNN	Checksum

Command 0x10: GetiPodFreeSpace

Direction: Accessory to Apple device

The accessory asks the Apple device to return the amount of free space on its storage system. The Apple device replies with RetiPodFreeSpace.

Table 3-264 GetiPodFreeSpace packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x10	Command ID: GetiPodFreeSpace
5	0xE2	Checksum

Command 0x11: RetiPodFreeSpace

Direction: Apple device to Accessory

The Apple device tells the accessory the current amount of free space (in bytes) in its storage system.

Table 3-265 RetiPodFreeSpace packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0A	Length of packet
3	0x0C	Lingo ID: Storage lingo

Byte number	Value	Comment
4	0x11	Command ID: RetiPodFreeSpace
5	0xNN	freeSpace (bits 63:56). The amount of the Apple device's free space.
6	0xNN	freeSpace (bits 55:48)
7	0xNN	freeSpace (bits 47:40)
8	0xNN	freeSpace (bits 39:32)
9	0xNN	freeSpace (bits 31:24)
10	0xNN	freeSpace (bits 23:16)
11	0xNN	freeSpace (bits 15:8)
12	0xNN	freeSpace (bits 7:0)
13	0xNN	Checksum

Command 0x12: OpeniPodFeatureFile

Direction: Accessory to Apple device

The accessory uses the `OpeniPodFeatureFile` command to open a feature file on the Apple device. [Table 3-266](#) (page 370) presents the format of the `OpeniPodFeatureFile` command packet.

In response, the Apple device returns a `RetiPodFileHandle` command with the feature file handle. If the feature type is not supported or the feature parameters are not valid, an `iPodACK` with bad parameter error status is returned instead of `RetiPodFileHandle`.

IMPORTANT: Feature files of different types can be open at the same time, but only one file of each type can be open. To open a second file of a given type, the first file must be closed.

Table 3-266 OpeniPodFeatureFile packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0xNN	Length of Packet
3	0x0C	Lingo ID: Storage
4	0x12	Command ID: OpeniPodFeatureFile
5	0xNN	featureType

Byte number	Value	Comment
6	0xNN	fileOptionsMask (bits 31:24)
7	0xNN	fileOptionsMask (bits 23:16)
8	0xNN	fileOptionsMask (bits 15:8)
9	0xNN	fileOptionsMask (bits 7:0)
10...N	0xNN	fileData (maximum 128 bytes) data to be appended at close. fileOptionsMask bit 0 must be set if this field is nonzero length.
(last byte)	0xNN	Checksum

Table 3-267 (page 371) lists the current possible `featureType` values for the `OpeniPodFeatureFile` command packet.

Table 3-267 `featureType` values

featureType	Description
0x00	Reserved
0x01	Radio tagging (no <code>fileOptionsMask</code> or <code>fileData</code> can be passed)
0x02	Cardio equipment workout; requires options to be set as specified in “ Recording Workout Data ” (page 573).
0x03-0xFF	Reserved

Table 3-268 (page 371) lists the current possible `fileOptionsMask` values for the `OpeniPodFeatureFile` command.

Table 3-268 `fileOptionsMask` values

fileOptionsMask	Description
bit 0	1 = On feature file close or accessory detach, append the <code>fileData</code> binary file data bytes to the file. Note that <code>fileData</code> must be nonzero length if this bit is set. 0 = Do not append any bytes to the file. <code>fileData</code> must be zero length if this bit is zero.
bit 1	1 = On file close or accessory detach, append the XML <code><ipodInfo></code> element to the file (includes <code><openTime></code> , <code><closeTime></code> , <code><model></code> , <code><softwareVersion></code> , and <code><serialNumber></code>). This option is applied before the bit 0 option. 0 = Do not write <code><ipodInfo></code> element to file. This option must be set when the XML Signature (bit 3) option is set = 1.
bit 2	Reserved (must be 0).

fileOptionsMask	Description
bit 3	1 = On file close or accessory detach, insert an XML <Signature> element to the file. This option is applied after bit 0 and bit 1 options. 0 = Do not insert an XML <Signature> element. When this option bit is set, the <ipodInfo> option (bit 1) must be set = 1.
bits 31:4	Reserved (must be 0).

Command 0x80: DeviceACK

Direction: Accessory to Apple device

The accessory must send the `DeviceACK` command to acknowledge the receipt of a Storage lingo command from the Apple device. The accessory must respond with bad parameter status whenever a Storage lingo command in the range 0x83 to 0xFF is received from the Apple device.

[Table 3-269](#) (page 372) shows the format of the `DeviceACK` command packet.

Table 3-269 DeviceACK packet

Byte number	Value	
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet
3	0x0C	Lingo ID: Storage
4	0x80	Command ID: DeviceACK
5	0xNN	ackStatus of command
6	0xNN	ID of Command being acknowledged
7	0xFF	File Handle for Command (set to 0xFF for not applicable)
8	0xNN	Checksum

[Table 3-270](#) (page 372) presents the possible values for the `ackStatus` byte.

Table 3-270 ackStatus values

ackStatus	Description
0x00	Success
0x01	N/A (reserved)

ackStatus	Description
0x02	Command failed
0x03	Out of resources
0x04	Bad parameter
0x05-0xFF	Reserved

Command 0x81: GetDeviceCaps

Direction: Apple device to Accessory

The `GetDeviceCaps` command may be sent by the Apple device to ask the accessory to return its storage capabilities (if any). [Table 3-271](#) (page 373) presents the format of the `GetDeviceCaps` command packet. The accessory replies with `RetDeviceCaps`.

Table 3-271 `GetDeviceCaps` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x02	Length of Packet
3	0x0C	Lingo ID: Storage
4	0x81	Command ID: <code>GetDeviceCaps</code>
5	0x71	Checksum

Command 0x82: RetDeviceCaps

Direction: Accessory to Apple device

The accessory must send the `RetDeviceCaps` command in response to a `GetDeviceCaps` command from the Apple device to determine the storage capabilities supported by the accessory. [Table 3-272](#) (page 373) presents the format of the `RetDeviceCaps` command packet.

Table 3-272 `RetDeviceCaps` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x18	Length of Packet

Byte number	Value	Comment
3	0x0C	Lingo ID: Storage
4	0x82	Command ID: RetDeviceCaps
5 - 23	0x00	Reserved (must be 0x00)
24	0xFF	Reserved (must be 0xFF)
25	0xNN	Major Storage lingo protocol version supported by the accessory (currently 0x01)
26	0xNN	Minor Storage lingo protocol version supported by the accessory (currently 0x02)
27	0xNN	Checksum

Lingo 0x0D: iPod Out Lingo

The iAP iPod Out lingo supports iPod Out mode, an Apple device operating mode described in ["iPod Out Mode"](#) (page 59).

Command History of the iPod Out Lingo

[Table 3-273](#) (page 374) shows the history of changes to the iPod Out lingo.

Table 3-273 iPod Out lingo command history

Lingo version	Command changes	Features
1.00	Add: 0x00–0x06	iPod Out support

iPod Out Lingo Commands

An accessory can use the iPod Out lingo commands to set up and manage iPod Out mode before or after the Apple device enters the mode. [Table 3-274](#) (page 374) summarizes the iPod Out lingo commands.

Table 3-274 iPod Out lingo command summary

Command	ID	Direction	Parameters
iPodACK	0x00	Dev to Acc	{ackStatus;1, origCmdID;1}
GetiPodOutOptions	0x01	Acc to Dev	{enabledOptions;1}
RetiPodOutOptions	0x02	Dev to Acc	{enabledOptions;1, optionsBits;4}
SetiPodOutOptions	0x03	Acc to Dev	{optionsBits;4}

Command	ID	Direction	Parameters
DevStateChangeEvent	0x04	Acc to Dev	{stateChange;1}
DevVideoScreenInfo	0x06	Acc to Dev	{totalScreenWidthInches;2, totalScreenHeightInches;2, totalScreenWidthPixels;2, totalScreenHeightPixels;2, iPodOutScreenWidthPixels;2, iPodOutScreenHeightPixels;2, screenFeaturesMask;1, screenGammaValue;1}

iPod Out Lingo Command Details

The following sections provide details of the iPod Out lingo commands.

Command 0x00: iPodACK

Direction: Apple device to Accessory

The Apple device sends this command to acknowledge the receipt of an iPod Out command from the attached accessory and report the status of its execution.

Table 3-275 iPodACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x0D	Lingo ID: iPod Out lingo
4	0x00	Command ID: iPodACK
5	0xNN	ackStatus: Status of command execution; see Table 3-276 (page 375).
6	0xNN	origCmdID: ID of the iPod Out command being acknowledged
7	0xNN	Checksum

Table 3-276 iPodACK command status values

Value	Meaning
0x00	Command executed successfully
0x01	Reserved

Value	Meaning
0x02	Command failed
0x03	Out of resources (Apple device's internal allocation failed)
0x04	Bad parameter (invalid command or input parameters)
0x05-0x06	Reserved
0x07	Not authenticated to use this lingo or command
0x08-0x0E	Reserved
0x0F	Command failed because operation timed out
0x10	Command unavailable in this Apple device mode
0x11-0xFF	Reserved

Command 0x01: GetiPodOutOptions

Direction: Accessory to Apple device

The accessory sends this command to query the capabilities of the Apple device while it is in iPod Out mode. The Apple device responds with a `RetiPodOutOptions` command. If the accessory passes 0x00 in the `enabledOptions` parameter, the Apple device reports all possible iPod Out options. If it passes 0x01 in `enabledOptions`, the Apple device reports only its currently set iPod Out options.

Table 3-277 GetiPodOutOptions packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x0D	Lingo ID: iPod Out lingo
4	0x01	Command ID: GetiPodOutOptions
5	0xNN	<code>enabledOptions</code> : 0x00 = report all iPod Out options the Apple device is capable of supporting; 0x01 = report only the iPod Out options currently set on the Apple device; 0x02-0xFF = reserved.
6	0xNN	Checksum

Command 0x02: RetiPodOutOptions

Direction: Apple device to Accessory

The Apple device sends this command to the accessory to report its iPod Out output capabilities, by setting bits in the big-endian `optionsBits` parameter. It also copies back the value of the `enabledOptions` parameter that the accessory sent in its `GetiPodOutOptions` command. If the value of `enabledOptions` is 0x01, only options that are currently set are reported.

Table 3-278 RetiPodOutOptions packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet
3	0x0D	Lingo ID: iPod Out lingo
4	0x02	Command ID: RetiPodOutOptions
5	0xNN	<code>enabledOptions</code> : 0x00 = reporting all iPod Out options the Apple device can support; 0x01 = reporting only the iPod Out options currently set on the Apple device; 0x02-0xFF = reserved.
6	0xNN	<code>optionsBits</code> (bits 31:24): Bits set to 1 to report iPod Out output options; all other bits are set to 0. See Table 3-279 (page 377).
7	0xNN	<code>optionsBits</code> (bits 23:16)
8	0xNN	<code>optionsBits</code> (bits 15:8)
9	0xNN	<code>optionsBits</code> (bits 7:0)
10	0xNN	Checksum

Table 3-279 Options for the iPod Out output from the Apple device

Bit	Meaning
00	Display audio content, including all media that don't have moving video, including songs, audio podcasts, audio books, and music videos played as songs only. <code>SetiPodOutOptions</code> cannot disable this option.
01	Display incoming phone call user interface.
02	Display SMS/MMS incoming text interface.
31:03	Reserved

Command 0x03: SetiPodOutOptions

Direction: Accessory to Apple device

The accessory sends this command to the iPod to limit the Apple device's iPod Out options. This lets the accessory control what content the user can ask it to display, in case there are legal regulations that the accessory needs to follow (for example, if it has a video screen visible to the driver of a car). The Apple device replies with an iPodACK command.

Table 3-280 SetiPodOutOptions packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x0D	Lingo ID: iPod Out lingo
4	0x03	Command ID: SetiPodOutOptions
5	0xNN	optionsBits (big-endian): Bits set to 1 to enable iPod Out output options; all other bits set to 0 to disable options. See Table 3-279 (page 377).
6	0xNN	optionsBits (bits 23:16)
7	0xNN	optionsBits (bits 15:8)
8	0xNN	optionsBits (bits 7:0)
9	0xNN	Checksum

Command 0x04: DevStateChangeEvent

Direction: Accessory to Apple device

The accessory sends this command to tell the Apple device that the accessory's state is changing. The Apple device responds by sending an iPodACK command, regardless of whether or not the Apple device acts on this information.

Table 3-281 DevStateChangeEvent packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x0D	Lingo ID: iPod Out lingo
4	0x04	Command ID: DevStateChangeEvent
5	0xNN	stateChange: Accessory state transition; see Table 3-282 (page 379).

Byte number	Value	Comment
6	0xNN	Checksum

Table 3-282 Accessory state transitions

Value	Meaning
0x00	Accessory display switching away from iPod Out
0x01	Accessory display switching back to iPod Out
0x02	Accessory audio switching away from iPod Out
0x03	Accessory audio switching back to iPod Out
0x04	Accessory entering “daytime” mode (full display brightness)
0x05	Accessory entering “nighttime” mode (display dimmed)
0x06-0xFF	Reserved

Command 0x06: DevVideoScreenInfo

Direction: Accessory to Apple device

This command passes the same parameters as the `ScreenInfoToken` sent during the IDPS process. Accessories can use it to dynamically change the screen size they allot to the Apple device after setting the token. The Apple device acknowledges it with an `iPodACK` command.

Table 3-283 DevVideoScreenInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x10	Length of packet
3	0x0D	Lingo ID: iPod Out lingo
4	0x06	Command ID: DevVideoScreenInfo
5	0xNN	<code>totalScreenWidthInches</code> (bits 15:8): Accessory’s approximate screen width in inches
6	0xNN	<code>totalScreenWidthInches</code> (bits 7:0)
7	0xNN	<code>totalScreenHeightInches</code> (bits 15:8): Accessory’s approximate screen height in inches
8	0xNN	<code>totalScreenHeightInches</code> (bits 7:0)

Byte number	Value	Comment
9	0xNN	totalScreenWidthPixels (bits 15:8): Number of pixels along the accessory's screen width
10	0xNN	totalScreenWidthPixels (bits 7:0)
11	0xNN	totalScreenHeightPixels (bits 15:8): Number of pixels along the accessory's screen height
12	0xNN	totalScreenHeightPixels (bits 7:0)
13	0xNN	iPodOutScreenWidthPixels (bits 15:8): Number of pixels along the iPod Out screen width allotment
14	0xNN	iPodOutScreenWidthPixels (bits 7:0)
15	0xNN	iPodOutScreenHeightPixels (bits 15:8): Number of pixels along the iPod Out screen height allotment
16	0xNN	iPodOutScreenHeightPixels (bits 7:0)
17	0xNN	screenFeaturesMask: A bitmask with bits set to represent the accessory screen's features; see Table 2-90 (page 139).
18	0xNN	screenGammaValue: A representation of the accessory display's gamma value. The Apple device obtains the gamma value from this byte by dividing it by 100 and adding 1.00. For example, a gamma of 2.200 is represented by a screenGammaValue of 120.
19	0xNN	Checksum

Lingo 0x0E: Location Lingo

The National Marine Electronics Association (NMEA) has established a standard interface for transferring global positioning system (GPS) information. The *NMEA 0183 Interface Standard*, Version 3.01 (released 2002/01), gives complete details about the sentence formats and contents.

The iAP Location lingo provides a mechanism by which NMEA sentences and other types of location information can be sent from accessories to attached Apple devices. Thus the Location lingo lets an accessory that generates raw location data send the data to an Apple device that collects, interprets, and displays it. Currently, only iOS 3.0 and later versions (including iOS 4) support the Location lingo.

Note: If an accessory provides or receives any kind of location data to or from an Apple device, it must use the Location lingo in addition to any other lingoes it supports.

Location Data Requirements

Accessories that generate NMEA GPS location sentences must support at least the GPGLA sentence type, as defined in the *0183 Interface Standard*. Apple devices also accept the GPRMC sentence type; if the accessory supports it, the accessory must provide it to the Apple device when using the Location lingo. The accessory may support additional NMEA sentence types, either standard or proprietary, and future Apple device models may also support more sentence types.

For maximum quality of the user experience, accessories that generate NMEA GPS location sentences must enable filtering and pass to the Apple device only sentence types that are in the filter list. Whenever possible, Apple devices try to conserve power by using location data from an external accessory instead of using on-board positioning technologies.

Power from the Apple Device for Accessories Using the Location Lingo

Attached accessories that support the Location lingo are notified of Apple device state changes, such as transitions between Power On and Hibernate. The accessory must keep its power consumption below the maximum allowed limits for each Apple device state; see Appendix A, “Apple Device Power States and Accessory Power,” in *MFi Accessory Hardware Specification*. Current on the Accessory Power line of the 30-pin connector will be off when the Apple device hibernates; when the Apple device wakes, the accessory must identify and authenticate itself, using the IDPS process specified in “[Accessory Signaling and Initialization](#)” (page 66).

Accessories that support the Location lingo may register for intermittent high power during their identification process. If so registered, they may draw up to 100 mA after receiving a `SetDevControl` command specifying GPS radio power on (see [Table 3-297](#) (page 393)). Accessories must reduce their power consumption to low power mode, as defined in the section “Accessory Power Policy” in *MFi Accessory Hardware Specification*, within 1 second of receiving a `SetDevControl` command specifying GPS radio power off. Regardless of the state of power set by the Location lingo, accessories must comply with Apple device state changes as detailed in Appendix A, “Apple Device Power States and Accessory Power,” in *MFi Accessory Hardware Specification*.

Command Summary

Certain prerequisites must be met for an accessory to use the Location lingo:

- The accessory must identify itself through the Identify Device Preferences and Settings (IDPS) process, as specified in “[Accessory Identification](#)” (page 519).
- To send and receive Location lingo and IDPS commands, the accessory must be able to generate and interpret transaction IDs, as described in “[Transaction IDs](#)” (page 525), throughout the communication session.
- The accessory must perform Authentication 2.0, as described in “[Authentication](#)” (page 71). Location lingo commands are usable after the accessory enters the background authentication state.

Note: With Apple devices that support wireless iAP over Bluetooth, an accessory need not be physically attached to an Apple device to send it location information.

The Apple device determines accessory capabilities by sending the Location lingo `GetDevCaps` command. A type parameter in this command specifies the type of capabilities queried. Capabilities type 0x00 requests the types of location services that the accessory supports and types 0x01–0x3F request details about the specific location services. Location capabilities include the following:

- Accessory general capabilities
- NMEA GPS location information (sentence filtering and GPGLL sentence generation are required).
- Location assistance information
- Asynchronous location notification control
- Capabilities specific to each location information type

Location lingo commands are extensible; new features, capabilities, and parameters may be added to future lingo versions. Lingo extensions will be made backward compatible so that existing implementations will continue to work. Accessory implementations must comply with the following rules to be compatible with future lingo versions:

- For all command responses, check for a packet length greater than or equal to the expected length. Do not check for an exact payload length, because this will fail after extensions are made. If the packet length is greater than expected, ignore any additional data.
- When reading lingo bitfield parameters, clear unrecognized bits to 0 before testing for feature bits.
- Latitude location data from an accessory must be within the range –90 to +90 degrees.
- Longitude location data from an accessory must be within the range –180 to +180 degrees.
- For other parameters, do range checks on their values and return a `DevACK` command with a bad parameter status (0x04) if a parameter is out of range.

Note: Apple devices and accessories that support the Location lingo must initialize their control states to empty or disabled when the device-accessory connection is first established. Initializing the control state is effectively the same as receiving `SetDevControl` commands for all `locType` values, with `ctlData` set to 0x0 for each. This will ensure that for all `locType` values, notifications are disabled by default and accessories are in low power mode, as defined in the section “Accessory Power Policy” in *MFi Accessory Hardware Specification*. One or more control states may be changed later, depending on the Apple device’s or accessory’s capabilities and usage requirements.

Table 3-284 (page 382) lists the Location lingo commands.

Table 3-284 Location lingo commands

ID	Name	Direction	Parameters
0x00	DevACK	Acc to Dev	{transID;2, cmdStatus;1, cmdIDOrig;1}
0x01	GetDevCaps	Dev to Acc	{transID;2, locType;1}

ID	Name	Direction	Parameters
0x02	RetDevCaps	Acc to Dev	{transID;2, locType;1, capsData;<var>}
0x03	GetDevControl	Dev to Acc	{transID;2, locType;1}
0x04	RetDevControl	Acc to Dev	{transID;2, locType;1, ctlData;<var>}
0x05	SetDevControl	Dev to Acc	{transID;2, locType;1, ctlData;<var>}
0x06	GetDevData	Dev to Acc	{transID;2, locType;1, dataType;1}
0x07	RetDevData	Acc to Dev	{transID;2, locType;1, dataType;1, sectCur;2, sectMax;2, locData;<var>}
0x08	SetDevData	Dev to Acc	{transID;2, locType;1, dataType;1, sectCur;2, sectMax;2, locData;<var>}
0x09	AsyncDevData	Acc to Dev	{transID;2, locType;1, dataType;1, sectCur;2, sectMax;2, locData;<var>}
0x0A–0x7F	Reserved		
0x80	iPodACK	Dev to Acc	{transID;2, cmdStatus;1, cmdIDOrig;1}
0x81–0xFF	Reserved		

A Typical Location Data Session

Table 3-285 (page 383) shows a typical exchange of commands by which an Apple device may obtain location data from an accessory. The first time such session could take place would be after the Apple device had acknowledged successful completion of the accessory identification process by sending `IDPSStatus` (see [Accessory Identification](#) (page 519)) and had begun the process of authenticating the accessory, as specified in "Authentication" (page 71).

Table 3-285 Sample command interchange

Step	Command	locType	dataType	Direction	Comments
1	GetDevCaps	0x00		Dev to Acc	The Apple device queries the accessory's system capabilities.
2	RetDevCaps	0x00		Acc to Dev	The accessory returns its system capabilities.
3	GetDevCaps	0x01		Dev to Acc	The Apple device queries the accessory's NMEA GPS location capabilities.
4	RetDevCaps	0x01		Acc to Dev	The accessory returns its NMEA GPS location capabilities.

Step	Command	locType	dataType	Direction	Comments
5	GetDevCaps	0x02		Dev to Acc	The Apple device queries the accessory's location assistance capabilities.
6	RetDevCaps	0x02		Acc to Dev	The accessory returns its location assistance capabilities.
7	GetDevControl	0x00		Dev to Acc	The Apple device queries the accessory's system control state.
8	RetDevControl	0x00		Acc to Dev	The accessory returns its system control state.
9	GetDevControl	0x01		Dev to Acc	The Apple device queries the accessory's NMEA GPS location control state.
10	RetDevControl	0x01		Acc to Dev	The accessory returns its NMEA GPS location control state.
11	SetDevControl	0x00		Dev to Acc	The Apple device sets the accessory's system control state for locType = 0x00 in accordance with the capabilities and controls sent to it by previous RetDevCaps and RetDevControl commands.
12	DevACK			Acc to Dev	The accessory acknowledges receipt of SetDevControl.
13	SetDevData	0x01	0x00	Dev to Acc	The Apple device sets the NMEA sentence filter list string.
14	DevACK			Acc to Dev	The accessory acknowledges receipt of SetDevData.
15	SetDevControl	0x01		Dev to Acc	The Apple device sets the accessory's system control state for locType = 0x01 in accordance with the capabilities and controls sent to it by previous RetDevCaps and RetDevControl commands.
16	DevACK			Acc to Dev	The accessory acknowledges receipt of SetDevControl.
17	GetDevData	0x02	0x03	Dev to Acc	The Apple device requests the satellite ephemeris maximum refresh interval.
18	RetDevData	0x02	0x03	Acc to Dev	The accessory returns the satellite ephemeris maximum refresh interval.

Step	Command	locType	dataType	Direction	Comments
19	GetDevData	0x02	0x04	Dev to Acc	The Apple device requests the satellite ephemeris recommended refresh interval.
20	RetDevData	0x02	0x04	Acc to Dev	The accessory returns the satellite ephemeris recommended refresh interval.
21	AsyncDevData	0x02	0x05	Acc to Dev	The accessory sends a request for current GPS time of the system.
22	iPodACK			Dev to Acc	The Apple device acknowledges receipt of the GPS time request.
23	AsyncDevData	0x02	0x02	Acc to Dev	The accessory sends an ephemeris URL to request an update to its ephemeris data.
24	iPodACK			Dev to Acc	The Apple device acknowledges receipt of the satellite ephemeris data URL string.
25	SetDevData	0x02	0x05	Dev to Acc	The Apple device sets the current GPS time of the accessory.
26	DevACK			Acc to Dev	The accessory acknowledges receipt of SetDevData.
27	SetDevData	0x02	0x00	Dev to Acc	The Apple device sets the current location data on the accessory for use in looking up the satellite position within the ephemeris.
28	DevACK			Acc to Dev	The accessory acknowledges receipt of SetDevData.
29	SetDevData	0x02	0x01	Dev to Acc	The Apple device sets the accessory's ephemeris with data downloaded from the URL provided in Step 23.
30	DevACK			Acc to Dev	The accessory acknowledges receipt of SetDevData.
If multisection data, repeat Steps 29–30 as needed. See "Multisection Data Transfers" (page 531).					
31	AsyncDevData	0x01	0x80	Acc to Dev	The accessory begins sending location data to the Apple device.
32	iPodACK			Dev to Acc	The Apple device acknowledges receipt of AsyncDevData.
If multisection data, repeat Steps 31–32 as needed. See "Multisection Data Transfers" (page 531).					

Command 0x00: DevACK

Direction: Accessory to Apple device

This command is sent by the accessory in response to a command sent from the Apple device. It has two forms:

- The Section form is sent if the command from the Apple device is part of a multisection data transfer. This form of the command is discussed in ["Multisection DevACK Command"](#) (page 532).
- The Default form, shown in [Table 3-286](#) (page 386), is sent under these conditions:
 - ❑ When a bad parameter is received, an unsupported or invalid command is received, or a command that does not return any data has completed.
 - ❑ To indicate completion of a multisection data transfer, as described in ["Multisection Data Transfers"](#) (page 531).

Table 3-286 Default DevACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x00	Command ID: DevACK
5	0xNN	transID [bits 15:8]: Transaction ID of command being acknowledged; see "Transaction IDs" (page 525)
6	0xNN	transID [bits 7:0]
7	0xNN	ackStatus: Status return; possible values are listed in Table 3-287 (page 386)
8	0xNN	cmdIDOrig: Original command ID for which this response is being sent.
9	0xNN	Checksum

Table 3-287 Valid Location lingo ackStatus values

Value	Meaning
0x00	Command OK
0x02	Command failed (valid command but did not succeed)
0x03	Out of resources (Apple device's internal allocation failed)
0x04	Bad parameter (invalid command or input parameters)

Value	Meaning
0x07	Not authenticated
0x0F	Command timeout
0x13	Section of multisection data transfer received successfully (see "Multisection Data Transfers" (page 531))

Command 0x01: GetDevCaps

Direction: Apple device to Accessory

This command is sent by the Apple device to get the accessory's capabilities. In response, the accessory sends a `RetDevCaps` command with the same `transactionID`, the capability type, and the requested capability data. If a specified `locType` is not supported, a `DevACK` must be returned with the bad parameter (0x04) status. Accessory support for capability type 0x00 (system caps) is mandatory; other capability types are optional. The Apple device requests the system caps from the accessory to determine its lingo version, system capabilities, and the other location types supported by the accessory. The accessory must send a corresponding `RetDevCaps` within 500 ms in response to the `GetDevCaps` command from the Apple device.

Table 3-288 GetDevCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x01	Command ID: GetDevCaps
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525)
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>locType</code> : The capability type; see Table 3-289 (page 387).
8	0xNN	Checksum

Table 3-289 Values for `locType`

Value	Meaning
0x00	System capabilities; see Table 3-291 (page 388).
0x01	NMEA GPS location capabilities; see Table 3-292 (page 389).
0x02	Location assistance capabilities; see Table 3-293 (page 390).

Value	Meaning
0x03–0x3F	Reserved
0x40–0xFF	Invalid

Command 0x02: RetDevCaps

Direction: Accessory to Apple device

The accessory sends this command in response to a `GetDevCaps` command from the Apple device, returning the transaction ID and the requested capabilities type. It informs the Apple device of its capabilities in `capsData`. If an accessory does not support a particular capability type, it must return a `DevACK` command with a bad parameter (0x04) status.

The `capsData` fields may be extended in the future as new features are added to the Location lingo. Apple devices receiving this command check for a minimum packet payload length (rather than an exact payload length) and parse only the supported fields, ignoring any additional data that may be appended.

Table 3-290 RetDevCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x02	Command ID: RetDevCaps
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The capability type; see Table 3-289 (page 387).
8–NN	0xNN	capsData: Capabilities data; see Table 3-291 (page 388), Table 3-292 (page 389), and Table 3-293 (page 390). The contents and length of this parameter depend on the value of locType.
NN+1	0xNN	Checksum

Table 3-291 System capabilities values (locType = 0x00)

Name	Size in bytes	Bits	Meaning
devVerMajor	1	Bits 7:0	Location lingo major version supported by the accessory

Name	Size in bytes	Bits	Meaning
devVerMinor	1	Bits 7:0	Location lingo minor version supported by the accessory
sysCapsMask	8	System capabilities (big-endian):	
		Bit 00	1 = Power management control support; Apple device-powered accessories must set this bit to allow power control by the Apple device.
		Bit 01	Reserved; set to 0.
		Bit 02	1 = Asynchronous location notification support; the accessory generates asynchronous location data and does not need to be polled. This bit must be set to 1.
		Bits 63:03	Reserved; set to 0.
locCapsMask	8	Location type support (big-endian):	
		Bit 00	1 = System support; must be set to 1.
		Bit 01	1 = NMEA GPS location support (see Note below)
		Bit 02	1 = Location assistance support; either this bit or bit 01 must be set
		Bits 63:03	Reserved; set to 0.

Note: Accessories that generate NMEA GPS location sentences must support at least the GPGLA sentence type. The Apple device also accepts the GPRMC sentence type; if the accessory supports it, the accessory must provide it to the Apple device when using the Location lingo. The accessory may support additional NMEA sentence types, either standard or proprietary. Sentences must be generated at intervals compliant with the *NMEA 0183 Interface Standard*, unless overridden by system controls or NMEA sentence filtering.

Table 3-292 NMEA GPS location capabilities values (locType = 0x01)

Name	Size in bytes	Bits	Meaning
nmeaGpsCaps	8	Bit 00 (big-endian)	Must be set to 1 to indicate NMEA GPS sentence filtering supported; the accessory must support a minimum filter list string length of 128 bytes.
		Bits 63:01	Reserved; set to 0.

Table 3-293 Location assistance capabilities values (locType = 0x02)

Name	Size in bytes	Bits	Meaning
locAsstData	8	Location assistance data types supported; the accessory's most recent RetDevCaps command must set the sysCapsMask capability bit 01 if one or more of the following bits are set (see Table 3-291 (page 388)):	
		Bit 00 (big-endian)	Reserved; set to 0.
		Bit 01	Satellite ephemeris data required, for faster location fix. This bit must be set if bit 02 is set.
		Bit 02	Satellite ephemeris data URL string (RFC 1738 compliant). This bit must be set if bit 01 is set. The accessory must supply a Web URL string for the Apple device to use to download ephemeris data and must accept that data when it is set by the Apple device (see Command 0x06: GetDevData (page 393) and Command 0x08: SetDevData (page 397)).
		Bit 03	Satellite position ephemeris data maximum refresh interval. If the accessory's long-term ephemeris degrades over time, this indicates the maximum amount of time during which downloaded data will be valid.
		Bit 04	Satellite position ephemeris data recommended refresh interval. If the accessory's long term ephemeris degrades over time, this indicates the recommended amount of time before a new download is needed.
		Bits 63:05	Reserved; set to 0.

Command 0x03: GetDevControl

Direction: Apple device to Accessory

This command is sent by the Apple device to get the accessory's control state. In response, the accessory must send a RetDevControl command within 500 ms, passing the same transaction ID and location type plus the accessory's control state information. A control type is supported only if the accessory's most recent RetDevCaps command set the locCapsMask capability bit. If the accessory receives an unsupported control type, it must return a DevACK command with a bad parameter (0x04) status.

Table 3-294 GetDevControl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x05	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x03	Command ID: GetDevControl
5	NN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525)
6	NN	transID [bits 7:0]
7	NN	locType: The location control type; see Table 3-289 (page 387).
8	NN	Checksum

Command 0x04: RetDevControl

Direction: Accessory to Apple device

This command is sent by the accessory in response to a `GetDevControl` command received from the Apple device. The transaction ID and control type received from `GetDevControl` must be returned with this command, along with the accessory's current control state of the specified control type. System control support is required for every accessory.

Note: The `RetDevControl` `ctlData` fields may be extended in the future as new features are added. Apple devices receiving this command check for a minimum packet payload length (rather than an exact payload length) and parse only the supported fields, ignoring any unrecognized additional data that may be appended.

Table 3-295 RetDevControl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0D	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x04	Command ID: RetDevControl
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The location control type; see Table 3-289 (page 387).

Byte number	Value	Comment
8–15	0xNNNNNNNNNNNNNNNN	ctlData: System control data, depending on the value of locType; see Table 3-297 (page 393).
16	0xNN	Checksum

Command 0x05: SetDevControl

Direction: Apple device to Accessory

This command is sent by the Apple device to set the accessory's control state. In response, the accessory must send a DevACK command within 500 ms, passing the same transaction ID and the status of the operation. Control types are valid only if the accessory's most recent RetDevCaps command set the associated RetDevCaps locCapsMask location type bits. If the Apple device tries to set reserved or unsupported control types or data, the accessory must return a DevACK command with the bad parameter (0x04) status.

Accessories that support the Location lingo may register for intermittent high power during their identification process. If so registered, they may draw up to 100 mA after receiving this command with a ctlData value specifying GPS radio power on; see [Table 3-297](#) (page 393). Accessories must reduce their power consumption to low power mode, as defined in the section "Accessory Power Policy" in *MFi Accessory Hardware Specification*, within 1 second of receiving this command with a ctlData value specifying GPS radio power off. Regardless of the state of power set by the Location lingo, accessories must comply with Apple device state changes as detailed in Appendix A, "Apple Device Power States and Accessory Power," in *MFi Accessory Hardware Specification*.

Note: The SetDevControl ctlData fields may be extended in the future as new features are added. Apple devices implementing the Location lingo check for a minimum packet payload length (rather than an exact payload length) and parse only the supported fields, ignoring any unrecognized additional data that may be appended.

Table 3-296 SetDevControl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0D	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x05	Command ID: SetDevControl
5	0xNN	transID [bits 15:8]: Transaction ID; see " Transaction IDs " (page 525)
6	0xNN	transID [bits 7:0]

Byte number	Value	Comment
7	0xNN	locType: The location control type; see Table 3-289 (page 387).
8–15	0xNNNNNNNNNNNNNNNN	ctlData: The accessory controls to be set; see Table 3-297 (page 393).
16	0xNN	Checksum

Table 3-297 ctlData values

locType	Name	Bits	Meaning
0x00	sysCtl	System control data:	
		Bits 01:00	Accessory GPS radio power and notify control state:
			00: Power off; if the accessory is powered solely by the Apple device, it must turn its GPS radio power off while this control state is set and reduce its current consumption to low power mode within 1 second of receiving SetDevControl.
			01–10: Reserved
			11: Power on; the accessory must turn its GPS radio power on when this control state is set. If the accessory is using Apple device power, it may draw up to 100 mA after receiving SetDevControl.
0x01	nmeaGpsCtrl	Bit 02	1 = Asynchronous location notifications enabled, 0 = notifications disabled.
		Bits 63:03	Reserved; set to 0.
0x01	nmeaGpsCtrl	Bit 00	When set to 1, NMEA GPS sentence filtering is enabled. When set to 0, NMEA GPS filtering is disabled; if asynchronous location notifications are enabled, all sentences must be passed to the Apple device.
		Bits 63:01	Reserved; set to 0.
0x02–0x3F	Reserved		
0x40–0xFF	Invalid		

Command 0x06: GetDevData

Direction: Apple device to Accessory

This command is sent by the Apple device to get location data of a specific type from the accessory. In response, the accessory sends a `RetDevData` command with the same transaction ID, location type, and the requested location data information. The accessory must send the corresponding `RetDevData` command within 500 ms after receiving a `GetDevData` command from the Apple device. In the case of multi-section responses, accessories must send each subsequent section within 500 ms of sending the previous section.

Table 3-298 GetDevData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x06	Command ID: <code>GetDevData</code>
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The capability type; see Table 3-289 (page 387).
8	0xNN	dataType: Location data type; See Table 3-299 (page 394).
9	0xNN	Checksum

Table 3-299 GetDevData dataType values

locType	Name	Data type	Meaning
0x00–0x01	Reserved		
0x02	locAsstData	Location assistance data; this data is available only if the accessory's most recent <code>RetDevCaps</code> command set the <code>locCapsMask</code> capability bit 01. The specific location data types are valid only if the associated <code>RetDevCaps</code> <code>locAssistance</code> bits were also set.	
		0x00–0x02	Reserved
		0x03	Satellite ephemeris data maximum required refresh interval. If the accessory's long-term ephemeris degrades over time, this indicates the maximum amount of time during which downloaded data will be valid.
		0x04	Satellite ephemeris data recommended refresh interval. If the accessory's long-term ephemeris degrades over time, this indicates the recommended amount of time before a new download is needed.

locType	Name	Data type	Meaning
		0x05–0xFF	Reserved
0x03–0x3F	Reserved		
0x40–0xFF	Invalid		

Command 0x07: RetDevData

Direction: Accessory to Apple device

This command is sent by the accessory in response to a `GetDevData` command received from the Apple device, returning its transaction ID, `locType`, and `dataType`, indicating the data type requested. The `locData` field is variable in length, based on the location data types listed in [Table 3-299](#) (page 394). The Apple device acknowledges `RetDevData` with an `iPodACK` command.

The Apple device has a maximum input data payload size of 500 bytes including the transaction ID. If the `locData` value sent by the accessory exceeds this size, it must be split into multiple sections as described in [Multisection Data Transfers](#) (page 531).

Table 3-300 RetDevData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x07	Command ID: RetDevData
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The capability type; see Table 3-301 (page 396).
8	0xNN	dataType: Location data type; see Table 3-301 (page 396).
9	0xNN	sectCur [bits 15:8]: Current payload section index (0x0000 = first or only section)
10	0xNN	sectCur [bits 7:0]
11	0xNN	sectMax [bits 15:8]: Maximum payload section index (0x0000 = only section, 0xNNNN = maximum or last section)
12	0xNN	sectMax [bits 7:0]

Byte number	Value	Comment
13	0xNN	totSize [bits 31:24]: Total size of locData in bytes. If multiple RetDevData packets are sent, the totSize field is included only in the first packet (sectCur = 0x0000).
14	0xNN	totSize [bits 23:16]
15	0xNN	totSize [bits 15:8]
16	0xNN	totSize [bits 7:0]
13–NN or 17–NN	<var>	locData: See Table 3-301 (page 396).
(last byte)	0xNN	Checksum

Table 3-301 RetDevData locData values

locType	Name	Meaning
0x00–0x01	Reserved	
0x02	locAsstData	Location assistance data: see Table 3-302 (page 396). This data type is valid only if the accessory's most recent RetDevCaps command set the sysCapsMask capability bit 01.
0x03–0x3F	Reserved	
0x40–0xFF	Invalid	

Table 3-302 locAsstData values (locType = 0x02) for RetDevData

dataType	Bytes	Field	Bytes	Subfield	Bytes	Value
0x00–0x02	Reserved.					
0x03	4	Satellite position ephemeris data maximum refresh interval, in milliseconds. Ephemeris data must be refreshed at least as often as this interval. This field must be sent in a single packet (sectCur = 0x0000 and sectMax = 0x0000).				
0x04	4	Satellite position ephemeris data recommended refresh interval, in milliseconds. Ephemeris data should be refreshed at least as often as this interval. This field must be sent in a single packet (sectCur = 0x0000 and sectMax = 0x0000). Its value must be less than or equal to the maximum refresh interval (dataType 0x03).				
0x05–0x3F	Reserved.					
0x40–0xFF	Invalid.					

Command 0x08: SetDevData

Direction: Apple device to Accessory

This command is sent by the Apple device to set location data on the accessory. In response, the accessory must return a `DevACK` command within 500 ms with the received transaction ID and the status of this operation. This time limit applies to both single and multisection responses. The `dataType` field represents the data type to be set; only certain data types, listed in [Table 3-304](#) (page 398), can be set. The `locData` field is variable in length, based on the location data types listed in [Table 3-306](#) (page 398).

The accessory should return its maximum payload size in its IDPS `accInfo` token; see [Table 2-82](#) (page 135). If the accessory does not specify a maximum payload size, the Apple device assumes a default size of 1024 bytes. If the `locData` value to be sent by the Apple device exceeds this size, it will be split into multiple sections as described in [Multisection Data Transfers](#) (page 531).

Table 3-303 SetDevData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x08	Command ID: SetDevData
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525)
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>locType</code> : The capability type; see Table 3-304 (page 398).
8	0xNN	<code>dataType</code> : the data types that can be set are listed in Table 3-304 (page 398).
9	0xNN	<code>sectCur</code> [bits 15:8]: Current payload section index (0x0000 = first or only section)
10	0xNN	<code>sectCur</code> [bits 7:0]
11	0xNN	<code>sectMax</code> [bits 15:8]: Maximum payload section index (0x0000 = only section, 0xNNNN = maximum or last section)
12	0xNN	<code>sectMax</code> [bits 7:0]
13	0xNN	<code>totSize</code> [bits 31:24]: Total size of <code>locData</code> in bytes. If multiple SetDevData packets are sent, the <code>totSize</code> field is included only in the first packet (<code>sectCur</code> = 0x0000).
14	0xNN	<code>totSize</code> [bits 23:16]
15	0xNN	<code>totSize</code> [bits 15:8]
16	0xNN	<code>totSize</code> [bits 7:0]

Byte number	Value	Comment
13– <i>NN</i> or 17– <i>NN</i>	< <i>var</i> >	locData: see Table 3-306 (page 398).
(last byte)	0x <i>NN</i>	Checksum

Table 3-304 Data types settable by SetDevData

locType	dataType	Description
0x01	0x00	NMEA sentence filter list string; see Table 3-305 (page 398).
0x02	0x00	Current point location data; see Table 3-306 (page 398).
0x02	0x01	Satellite ephemeris data; see Table 3-306 (page 398).
0x02	0x05	Current GPS time on accessory; see Table 3-306 (page 398).

Table 3-305 nmeaGpsLocData values (locType = 0x01)

dataType	Value
0x00	NMEA sentence filter list string, a comma-delimited set of NMEA standard and/or proprietary uppercase acronyms for sentence types. The accessory must support a filter string of at least 128 bytes. Depending on the string length, the Apple device may split this command into multiple sections. Only the last section will include the string null terminator. An example sentence list string could be GPGGA , GPRMC, specifying the two sentence types to be enabled. The accessory must not send sentence types not in the filter list. A string consisting of only the null terminator disables the sending of all NMEA sentences by the accessory.
0x01–0xFF	Reserved

Table 3-306 locAsstData values (locType = 0x02) for SetDevData

dataType	Bytes	Field	Bytes	Subfield	Bytes	Value
0x00	16	Current point location data; must be sent as a single packet (sectCur = 0x0000 and sectMax = 0x0000). See Note , below.				
		locWeek	2	GPS Week number. The 13 least significant bits comprise a modulo-8192 representation of the current GPS Week number, as defined by IS-GPS-200 PIRN-002, Section 30.3.3.1.1.1. The GPS Week Number count began at UTC midnight Saturday-Sunday January 5-6, 1980. Since then the count has been broadcast as part of the GPS message and has incremented by 1 each UTC Saturday-to-Sunday transition. The GPS Week number will roll over in the year 2137. The week containing Tuesday, April 28, 2009 is GPS Week 1,529.		

dataType	Bytes	Field	Bytes	Subfield	Bytes	Value
		locTime	4	GPS Time of Week. The 30 least significant bits represent the time when the location point was determined. This is measured as milliseconds in the week from the last UTC Saturday-to-Sunday transition, a number between 0 and 604,800,000.		
		locPoint	8	Current location point in millionths of a degree latitude and longitude		
				latDegArc	4	Signed 32-bit latitude: 0x00000000 = ±0° (Equator), 0x055D4A80 = +90° (maximum north), 0xFAA2B580 = −90° (maximum south).
				lonDegArc	4	Signed 32-bit longitude: 0x00000000 = ±0° (Greenwich meridian), 0x0ABA9500 = +180.000000° (maximum east), 0xF5456B01 = −179.999999° (maximum west).
		locAccuracy	2	Location accuracy radius		
				locRadius	2	Point location accuracy radius in meters; a smaller radius number means a more accurate point location
0x01	<var>	ephData	<var>	Satellite ephemeris data. This data does not have a standard format and is not parsed by the Location lingo.		
0x02–0x04		Reserved.				
0x05	6	The Apple device's current GPS time.				
		locWeek	2	GPS Week number. The 13 least significant bits comprise a modulo-8192 representation of the Apple device's current GPS Week number, as defined by IS-GPS-200 PIRN-002, Section 30.3.3.1.1.1. The GPS Week Number count began at UTC midnight Saturday-Sunday January 5-6, 1980. Since then the count has been broadcast as part of the GPS message and has incremented by 1 each UTC Saturday-to-Sunday transition. The GPS Week number will roll over in the year 2137. The week containing Tuesday, April 28, 2009 is GPS Week 1,529.		
		locTime	4	GPS Time of Week. The 30 least significant bits represent the Apple device's current GPS time. This is measured as milliseconds in the week from the last UTC Saturday-to-Sunday transition, a number between 0 and 604,800,000.		
0x06–0x3F		Reserved.				
0x40–0xFF		Invalid.				

Note: The Apple device pushes system GPS time and point location data to the accessory, to be used in conjunction with ephemeris data. The GPS time of the Apple device's point location data may be significantly behind system time, because it represents the last location lock that the Apple device was able to achieve. When calculating its reference into ephemeris data, the accessory should consider both the system time and the point location time to validate the Apple device's data.

The Apple device pushes system time to the accessory when launching location services or in response to accessory inquiries. Once ephemeris data has been requested the Apple device continues to push point location data asynchronously, as the data accuracy changes. It also delivers updated ephemeris data periodically, as available, after the accessory makes its initial request.

Command 0x09: AsyncDevData

Direction: Accessory to Apple device

This command is sent by the accessory to notify the Apple device that location data is available. It is valid only if the accessory supports asynchronous notifications (see [Table 3-297](#) (page 393)) and the notification category has been enabled by a `SetDevControl` command. The `dataType` field represents the data type being sent; the currently valid types are listed in [Table 3-308](#) (page 401). The `locData` field is variable in length, depending on the data being sent.

No iAP command may exceed a value of 500 bytes in its length-of-packet-payload field. If the `locData` value sent by the accessory exceeds this size, it must be split into multiple sections with the same transaction ID, as described in [Multisection Data Transfers](#) (page 531). The Apple device acknowledges each `AsyncDevData` packet with an `iPodACK` command.

The accessory may send an `AsyncDevData` command with a `dataType` of 0x02 or 0x05 any time Accessory Power is high (pin 13 of the 30-pin connector), regardless of the Apple device's control state as set by `SetDevControl`. The Apple device responds immediately with an `iPodACK` command, but the data requested will not be sent until the Apple device's location services are actually engaged by an application. The accessory must respond to other iAP commands while waiting for a response. The URL must not be sent to the Apple device more often than the maximum ephemeris refresh interval established by the accessory via `RetDevData`, and in any case not more often than once every 5 minutes.

Table 3-307 AsyncDevData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x09	Command ID: AsyncDevData
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see "Transaction IDs" (page 525)
6	0xNN	<code>transID</code> [bits 7:0]

Byte number	Value	Comment
7	0xNN	locType: The capability type; see Table 3-308 (page 401).
8	0xNN	dataType: Location data type; see Table 3-308 (page 401).
9	0xNN	sectCur [bits 15:8]: Current payload section index (0x0000 = first or only section)
10	0xNN	sectCur [bits 7:0]
11	0xNN	sectMax [bits 15:8]: Maximum payload section index (0x0000 = only section, 0xNNNN = maximum or last section)
12	0xNN	sectMax [bits 7:0]
13	0xNN	totSize [bits 31:24]: Total size of locData in bytes. If multiple AsyncDevData packets are sent, the totSize field is included only in the first packet (sectCur = 0x0000). See Multisection Data Transfers (page 531).
14	0xNN	totSize [bits 23:16]
15	0xNN	totSize [bits 15:8]
16	0xNN	totSize [bits 7:0]
13–NN or 17–NN	<var>	locData: Data sent by AsyncDevData; see Table 3-308 (page 401).
(last byte)	0xNN	Checksum

Table 3-308 locData values that can be sent by AsyncDevData

locType	dataType	locData value
0x01	0x80	String of NMEA 0183 compliant sentences. Each sentence must be sent as a full NMEA 0183 compliant sentence including the \$GP, \$P, or !P prefix, the *NN checksum, and the <CR><LF> suffix.
0x02	0x02	Null-terminated URL string (RFC 1738 compliant) to a web site for satellite ephemeris data (such as http://www.yourcompany.com/long-term-ephemeris). This must be a full URL, including http:// . Only http and https transfer protocols are currently supported.
0x02	0x05	Request for current system time in GPS time (GPS Week number and milliseconds offset into the week; see Table 3-306 (page 398)).

Command 0x80: iPodACK

Direction: Apple device to Accessory

This command is sent by the Apple device in response to a command sent from the accessory. It has two forms:

- The Default form, shown in [Table 3-309](#) (page 402), is sent under these conditions:
 - When a bad parameter is received, an unsupported or invalid command is received, or a command that does not return any data has completed.
 - To indicate completion of a multisection data transfer, as described in ["Multisection Data Transfers"](#) (page 531).
- The Section form is sent if the command from the accessory is part of a multisection data transfer. This form of the command is discussed in ["Multisection iPodACK Command"](#) (page 532).

Table 3-309 Default iPodACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x80	Command ID: iPodACK
5	0xNN	transID [bits 15:8]: Transaction ID of command being acknowledged; see "Transaction IDs" (page 525)
6	0xNN	transID [bits 7:0]
7	0xNN	ackStatus: Status return; possible values are listed in Table 3-287 (page 386)
8	0xNN	cmdIDorig: Original command ID for which this response is being sent.
9	0xNN	Checksum

Sample Identification Sequences

This section provides the following commented listings of sample command sequences in which an accessory identifies itself to an Apple device:

- [Table 3-310](#) (page 403) lists sample commands for an accessory that supports the General and Extended Interface lingoes. For Extended Interface command details, see ["The Extended Interface Protocol"](#) (page 409).
- [Table 3-311](#) (page 404) lists sample commands for an accessory that supports the General, Simple Remote, and Display Remote lingoes.
- [Table 3-312](#) (page 406) lists sample commands for an accessory that supports the General, Simple Remote, Display Remote, and Digital Audio lingoes.

Table 3-310 Identification of lingoes 0x00+0x04

Step	Accessory command	Apple device command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> ■ If the accessory receives no ACK command within 500 ms, it may retry Step 1 at 500 ms intervals as long as Apple Device Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them. ■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the Apple device does not support IDPS. Within 800 ms the accessory should send an IdentifyDeviceLingoes command with all fields set to 0xFF. This causes the Apple device to display an "Accessory not supported" message. <p>Alternatively, an accessory receiving a status of 0x04 may send an IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). A sample command sequence is listed in Table F-43 (page 623).</p> <ul style="list-style-type: none"> ■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1. 			
3	SetFIDTokenValues		setting 11 FID tokens ; IdentifyToken = (lingoes: 0/4 options 0x00000002 device ID 0x00000200); AccCapsToken = 0x00000000000000205; AccInfoToken = Acc name (Apple); AccInfoToken = Acc FW version (v9.8.7); AccInfoToken = Acc HW version (v1.2.3); AccInfoToken = Acc manufacturer (Apple Inc.); AccInfoToken = Acc model number (ModelNumber-XYZ); EAPProtocolToken = 1 (com.Apple.ProtocolMain); EAPProtocolToken = 2 (com.Apple.ProtocolAlt); iPodPreferenceToken = (setting 'off' for preference class 'video out setting' with restore on exit 'not selected'); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'not selected')

Step	Accessory command	Apple device command	Comment
4		RetFIDTokenValueACKs	11 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; EAPProtocolToken = (1) accepted; EAPProtocolToken = (2) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (line out usage) accepted
5	EndIDPS		status 'finished with IDPS; proceed to authentication'
6		IDPSStatus	status 'ready for auth'
7		GetDevAuthentication-Info	no params

Table 3-311 Identification of lingo 0x00+0x02+0x03

Step	Accessory command	Apple device command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'

If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,

- If the accessory receives no ACK command within 500 ms, it may retry Step 1 at 500 ms intervals as long as Apple Device Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them.

- If the accessory receives an ACK status of 0x04 (Bad Parameter), the Apple device does not support IDPS. Within 800 ms the accessory should send an IdentifyDeviceLingo command with all fields set to 0xFF. This causes the Apple device to display an "Accessory not supported" message.

Alternatively, an accessory receiving a status of 0x04 may send an IdentifyDeviceLingo command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See ["Cancelling a Current Authentication Process With IdentifyDeviceLingo"](#) (page 99). A sample command sequence is listed in [Table F-44](#) (page 628).

- If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1.

Step	Accessory command	Apple device command	Comment
3	SetFIDTokenValues		setting 11 FID tokens ; IdentifyToken = (lingoes: 0/2/3 options 0x00000002 device ID 0x00000200); AccCapsToken = 0x00000000000000205; AccInfoToken = Acc name (Apple); AccInfoToken = Acc FW version (v9.8.7); AccInfoToken = Acc HW version (v1.2.3); AccInfoToken = Acc manufacturer (Apple Inc.); AccInfoToken = Acc model number (ModelNumber-XYZ); EAPProtocolToken = 1 (com.Apple.ProtocolMain); EAPProtocolToken = 2 (com.Apple.ProtocolAlt); iPodPreferenceToken = (setting 'off' for preference class 'video out setting' with restore on exit 'not selected'); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'not selected')
4		RetFIDTokenValueACKs	11 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; EAPProtocolToken = (1) accepted; EAPProtocolToken = (2) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (line out usage) accepted
5	EndIDPS		status 'finished with IDPS; proceed to authentication'
6		IDPSStatus	status 'ready for auth'
7		GetDevAuthentication-Info	no params

Table 3-312 Identification of lingoes 0x00+0x02+0x03+0x0A

Step	Accessory command	Apple device command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> ■ If the accessory receives no ACK command within 500 ms, it may retry Step 1 at 500 ms intervals as long as Apple Device Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them. ■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the Apple device does not support IDPS. Within 800 ms the accessory should send an IdentifyDeviceLingoes command with all fields set to 0xFF. This causes the Apple device to display an "Accessory not supported" message. <p>Alternatively, an accessory receiving a status of 0x04 may send an IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). A sample command sequence is listed in Table F-45 (page 631).</p> <ul style="list-style-type: none"> ■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1. 			
3	SetFIDTokenValues		setting 11 FID tokens ; IdentifyToken = (lingoes: 0/2/3/10 options 0x00000002 device ID 0x00000200); AccCapsToken = 0x00000000000000215; AccInfoToken = Acc name (Apple); AccInfoToken = Acc FW version (v9.8.7); AccInfoToken = Acc HW version (v1.2.3); AccInfoToken = Acc manufacturer (Apple Inc.); AccInfoToken = Acc model number (ModelNumber-XYZ); EAPProtocolToken = 1 (com.Apple.ProtocolMain); EAPProtocolToken = 2 (com.Apple.ProtocolAlt); iPodPreferenceToken = (setting 'off' for preference class 'video out setting' with restore on exit 'not selected'); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'not selected')

Step	Accessory command	Apple device command	Comment
4		RetFIDTokenValueACKs	11 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; EAPProtocolToken = (1) accepted; EAPProtocolToken = (2) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (line out usage) accepted
5	EndIDPS		status 'finished with IDPS; proceed to authentication'
6		IDPSStatus	status 'ready for auth'
7		GetDevAuthentication- Info	no params
8	RetDevAuthentication- Info		returns authentication version and X.509 certificate data
9		AckDevAuthentication- Info	acknowledging 'auth info supported'
10		GetAccSampleRateCaps	no params
11		GetDevAuthentication- Signature	sends authentication challenge
12	RetAccSampleRateCaps		returning sample rates '8000 11025 12000 16000 22050 24000 32000 44100 48000'
13	RetDevAuthentication- Signature		returns digital signature in response to authentication challenge
14		TrackNewAudio- Attributes	sample rate 44100; Sound Check value 0; track volume adjustment 0
15	AccAck		acknowledging 'TrackNewAudioAttributes'

Step	Accessory command	Apple device command	Comment
16		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
17		TrackNewAudio-Attributes	sample rate 44100; Sound Check value 0; track volume adjustment 0
18	AccAck		acknowledging 'TrackNewAudioAttributes'
19		TrackNewAudio-Attributes	sample rate 44100; Sound Check value 0; track volume adjustment 0
20	AccAck		acknowledging 'TrackNewAudioAttributes'

The Extended Interface Protocol

The Extended Interface protocol allows the user interface of an Apple devices to be translated to other environments; for example, a vehicle entertainment system such as a dashboard radio, large-screen display, or steering wheel mounted remote control. This document specifies command packets that allow access to the Apple devices database and device management over serial or USB connections. This interface has the following advantages:

- It allows file-system and database-format independence for the interfacing body.
- It allows the interfacing body to remain ignorant of the encoders and decoders used to process digital audio information.

Note: 32- and 16-bit quantities are sent in big-endian form, as the bit breakdowns in this specification show.

Major Apple Device Operating Modes

For the purpose of this document, Apple devices can be considered to operate in two major modes: Standard UI mode and Extended Interface mode. In addition there is the Sleep state, which can overlay the major modes in specific circumstances. For more information on Apple device power states, see *MFfi Accessory Hardware Specification*.

Standard UI Mode is the user interface mode that allows an Apple device to be driven by its front panel display and buttons. Alternatively, the Apple device may be placed in Extended Interface mode and controlled by an accessory, using the Extended Interface protocol and its lingo (Lingo 0x04) as described in this document. The present chapter briefly summarizes the Extended Interface and Sleep modes.

Extended Interface Mode

The Apple device transitions into the Extended Interface mode when an accessory is connected to it and issues an `EnterRemoteUIMode` command. The transition to Extended Interface mode is described in "[Accessory Identification and Authentication](#)" (page 412).

If the Apple device is playing an audio track during this transition, the playback is automatically paused; video tracks are stopped. By default, the Apple device's display changes to an "OK to disconnect" screen such as one of those shown in [Figure 4-1](#) (page 410).

Figure 4-1 Typical “OK to disconnect” screens

The Extended Interface protocol allows accessories to replace the checkmark graphic with a downloaded image set through "[Command 0x0032: SetDisplayImage](#)" (page 485). Removing power from the Apple device while a connection remains results in the Apple device going into a Sleep state after two minutes of inactivity. The keys and wheel on the front of the Apple device are disabled when in Extended Interface mode.

The Apple device transitions back to Standard UI mode when any of the following occurs:

- The accessory issues an `ExitRemoteUIMode` command.
- The accessory reidentifies itself, using the `StartIDPS` command (see "[Accessory Signaling and Initialization](#)" (page 66)).
- The accessory is disconnected from the Apple device.

If the Apple device is playing a track during this transition, the playback is automatically paused. Any Apple device settings with the restore on exit feature set are restored when the Apple device is disconnected.

Sleep State

The Apple device's screen, playback, and most major parts of the Apple device are off while the Apple device is in the Sleep state. The Apple device transitions from Extended Interface mode to the Sleep state when power is detached and playback is idle. A two minute period of inactivity is required before the Apple device transitions into the Sleep state. When power is restored, the Apple device returns to the Extended Interface mode.

Note: iOS devices do not have a sleep state.

An Apple device will not sleep while it remains attached to an active USB host. The USB host must switch off its host controller to force an Apple device in Extended Interface mode into the Sleep state. If the Apple device is not currently in Extended Interface mode, playback must be paused before the host controller is turned off. In Extended Interface mode, there is no need to pause Apple device playback before turning off the host controller because this action generates a disconnect event that causes the Apple device to exit the Extended Interface mode and allows the Apple device to transition into the Sleep state. Attaching USB power to an Apple device in Sleep state will wake it up.

Refer to the *MFi Accessory Hardware Specification* for information about transitions to the Sleep state from Standard UI mode.

Packet Formats

The Extended Interface packet format is compatible with the packet formats described in "Command Packet Formats" (page 76). The primary difference between the two is the use of a 16-bit command ID field in Lingo 0x04 packets versus an 8-bit command ID field in other packet-based lingoes. Depending on the length of the packet, either the small packet format (payloads of 252 bytes or less) or the large packet format (253–65532 byte payloads) will be used.

Note: The Extended Interface start of packet (SOP) byte has the same value as the start of packet byte used in other iAP packets.

Small Packet Format

For packets whose payloads are 252 bytes or less, use the small packet format. The small packet format is shown in [Table 4-1](#) (page 411).

Table 4-1 Small packet format

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0xNN	Command ID (bits 15:8)
5	0xNN	Command ID (bits 7:0)
6...NN	0xNN	Packet command data (0–252 bytes)
(last byte)	0xNN	Packet payload checksum byte

Large Packet Format

For packets whose payloads are between 253 bytes and 65532 bytes in length, use the large packet format. The large packet format is shown in [Table 4-2](#) (page 412).

Table 4-2 Large packet format

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x00	Packet payload length marker
3	0xNN	Packet payload length (bits 15:0)
4	0xNN	Packet payload length (bits 7:0)
5	0x04	Lingo ID: Extended Interface lingo
6	0xNN	Command ID (bits 15:8)
7	0xNN	Command ID (bits 7:0)
8...NN	0xNN	Packet command data (0–65532 bytes)
(last byte)	0xNN	Packet payload checksum byte

Packet Details

The sync byte (0xFF) is not considered part of the packet. It is sent merely to facilitate automatic baud rate detection and correction when communicating over the UART serial port link. The sync byte is required only when communicating over the UART serial port link. It must not be sent when communicating over the USB port link. The packet payload length is the number of bytes in the packet not including the sync byte, packet start byte, packet payload length byte or bytes, or packet payload checksum. That is, it is the length of the lingo ID, the command ID, and the command data, if any. The lingo ID specifies the category this communication falls under; in this case, it is the Extended Interface (0x04). The command ID is a 16-bit value, as compared with 8-bit command ID used by many other lingoes, that gives a specific indication of the packet's purpose.

The sum of all the bytes from the packet payload length (or length marker, if applicable) through the packet payload checksum is 0. The checksum must be calculated appropriately, by adding the bytes together as signed 8-bit values, discarding any signed 8-bit overflow, and negating the sum to create the signed 8-bit checksum byte.

Accessory Identification and Authentication

When attached, an accessory must detect the Apple device and then identify and authenticate itself, as described in "[Accessory Signaling and Initialization](#)" (page 66). To do this, the accessory must send a `StartIDPS` command to begin the Identify Device Preferences and Settings (IDPS) process.

As part of its identification and authentication, the accessory should query the Apple device's support for Extended Interface mode by sending a `GetiPodOptionsForLingo` command for Lingo 0x04. The 64-bit field in the `RetiPodOptionsForLingo` reply declares which Extended Interface features that Apple devices support, as listed in [Table 4-3](#) (page 413).

Table 4-3 `RetiPodOptionsForLingo` option bits for Lingo 0x04

Bit	Description
0	Video browsing
1	The following Extended Interface commands are enabled: <code>GetDBiTunesInfo</code> <code>RetDBiTunesInfo</code> <code>GetUIDTrackInfo</code> <code>RetUIDTrackInfo</code> <code>GetDBTrackInfo</code> <code>RetDBTrackInfo</code> <code>GetPBTrackInfo</code> <code>RetPBTrackInfo</code>
2	Nested playlists
3	Reserved
4	The Apple device displays images sent by " Command 0x0032: SetDisplayImage " (page 485)
63:5	Reserved

A simplified sequence of iAP General lingo commands for accessory identification and authentication is shown in [Table 4-4](#) (page 413). These commands identify the accessory, determine which options the attached Apple device supports, set the accessory's preferences in the Apple device, and authenticate the accessory.

Table 4-4 Simplified IDPS and authentication command sequence

Step	Accessory command	Apple device command	Comment
1	<code>StartIDPS</code>		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'

Step	Accessory command	Apple device command	Comment
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> ■ If the accessory receives no ACK command within 500 ms, it may retry Step 1 at 500 ms intervals as long as Apple Device Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them. ■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the Apple device does not support IDPS. Within 800 ms the accessory must send an IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. ■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1. 			
3	GetiPodOptions-ForLingo		getting options for General Lingo
4		RetiPodOptions-ForLingo	returning options for General Lingo
5	GetiPodOptions-ForLingo		getting options for Extended Interface Lingo
6		RetiPodOptions-ForLingo	returning options for Extended Interface Lingo
7	SetFIDTokenValues		setting full ID (FID) tokens that identify accessory preferences
8		RetFIDTokenValueACKs	acknowledging FID tokens that identify accessory preferences
Repeat Steps 7 and 8 until all required accessory preferences have been set and acknowledged			
9	EndIDPS		status 'finished with IDPS; proceed to authentication'
10		IDPSStatus	status 'ready for auth'
11		GetDevAuthentication-Info	no params
12	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1; cert data: ...
13		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
14	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1; cert data: ...

Step	Accessory command	Apple device command	Comment
15		AckDevAuthentication-Info	acknowledging 'auth info supported'
16	EnterRemoteUIMode		entering Extended Interface mode
17		ACK	acknowledging 'Command Pending' to command 'General Lingo::EnterRemoteUIMode'
18		ACK	acknowledging 'Success (OK)' to command 'General Lingo::EnterRemoteUIMode'
19		GetDevAuthentication-Signature	offering challenge '...' with retry counter 1
20	RetDevAuthentication-Signature		returning signature '...'
21		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'

Note: The process of accessory identification is different for accessories that must support the 3G iPod. See “Interfacing With the 3G iPod” in *MFi Accessory Hardware Specification*.

Command Transports

To establish reliable communication over UART transport, it may be necessary to repeat the accessory identification sequence multiple times. There can be problems when the Apple device tries to synchronize with the accessory's baud rate, resulting in packet loss. Repeating the accessory identification sequence until it is successful ensures that the Apple device identifies the accessory correctly. This problem does not occur with USB transport.

Accessories using a UART serial port link should send an extra sync byte before a command packet in order to wake up a sleeping Apple device. This recommendation does not apply while the Apple device is in Extended Interface mode because Apple devices do not sleep in this mode as long as external power is supplied by an accessory. If the accessory allows the Apple device to transition to Sleep state by removing external power, it should send extra sync bytes when waking up the Apple device. For further information about supported transport protocol links, see *MFi Accessory Hardware Specification*.

Entering Extended Interface Mode

An Apple device does not automatically enter into Extended Interface mode; the accessory must explicitly switch into and out of the Extended Interface mode. To confirm that the Apple device has gone into Extended Interface mode, the accessory sends it the Lingo 0x04 command `GetPlayStatus`. Receiving a valid

`ReturnPlayStatus` command tells the accessory that the Apple device has entered Extended Interface mode and Lingo 0x04 commands can be used. The Apple device will return Lingo 0x04 ACK commands with a bad parameter status if the accessory is not communicating properly with the Apple device in Extended Interface mode.

Accessories may send Extended Interface command packets only after they have successfully identified and switched into Extended Interface mode. Accessories must still obey the other inter-packet and inter-command timing issues specified in ["Command Timings"](#) (page 184).

Using the Extended Interface Protocol

There are four General lingo (Lingo 0x00) commands that allow accessories to determine what mode an Apple device is in and to switch between the two major modes, Standard UI and Extended Interface. These commands were implemented to allow a accessory to switch between modes without having to unplug the accessory. Multilingo accessories must use these commands to switch into and out of the Extended Interface mode.

Note: Multilingo accessories that support the Extended Interface do not automatically switch into the Extended Interface mode after the identification process completes. These accessories must use the General lingo mode switching commands to explicitly switch into Extended Interface mode.

[Figure 4-4](#) (page 430) lists the General lingo command codes for querying, entering, and exiting the Extended Interface protocol.

Table 4-5 General lingo commands for switching modes

Command ID	General lingo command	Protocol version	Requires authentication	
			UART serial port link	USB port link
0x03	<code>RequestRemoteUIMode</code>	1.00	No	Yes
0x04	<code>ReturnRemoteUIMode</code>	1.00	No	Yes
0x05	<code>EnterRemoteUIMode</code>	1.00	No	Yes
0x06	<code>ExitRemoteUIMode</code>	1.00	No	Yes

Two logical entities need to be managed while browsing and playing content in Extended Interface mode: the content Database Engine and the Playback Engine. The following sections describe those engines and give an example of command traffic between an accessory in Extended Interface mode and an Apple device.

[Table 4-10](#) (page 436) lists the complete set of Extended Interface command IDs, noting which are relevant to the Database and Playback Engines, as well as which the protocol versions in which those commands were introduced. Unless otherwise noted, all subsequent protocol versions will support existing commands.

The Playback Engine

The Playback Engine is active when an Apple device is in a playback state, such as play, fast forward, and rewind. It has a special play list, called the Now Playing playlist, that is used to determine what track or content item will be played next. The `PlayCurrentSelection` command is commonly used to transfer the currently selected database items to the Now Playing playlist and start the player at a specified item within that list. The `SelectDBRecord` command, with a track, audiobook, or GeniusMix category and a valid index, also transfers selected database items to the Now Playing playlist. Changes to the database selection before or after this command have no effect on the current playback.

Playback Behavior

An Apple device's behavior when it receives a `PlayCurrentSelection` packet may depend on the shuffle setting, the repeat setting, and on the kind of tracks in a playlist. Consider a playlist with the following tracks:

Track A
Track B
Track C
Track D
Track E

If shuffle mode is on, a `PlayCurrentSelection` packet with an index of 0 or greater causes the playlist to be shuffled randomly but with the first track set to the selected index. For example, with an index of 4, the playlist may be sent to the Playback Engine in the following order and play starts with Track E. If repeat is set to One, track E will repeat; if it is set to All, the entire playlist will repeat.

Track E
Track B
Track A
Track D
Track C

Note: The index numbers of tracks in the Playback Engine are always arranged in the current playlist order, starting with 0.

If shuffle mode is off, a `PlayCurrentSelection` packet with an index of 0 or greater causes the playlist to be sent to the Playback Engine in the current playlist order but with the first track set to the passed index. With an index of 2, using the current example, the playlist is sent to the Playback Engine in this order but play starts at Track C, proceeds to Track D, and so on:

Track A
Track B
Track C
Track D
Track E

Note: Shuffle mode affects the playing of podcasts only if the user has disabled their Skip When Shuffle attribute in iTunes before syncing the Apple device.

If Repeat is set to 0, playback will stop at the end of the playlist. If Repeat is set to 1, the track corresponding to the selected track index will repeat. If Repeat is set to All, the entire playlist will repeat.

In all the foregoing scenarios, the Apple device returns a successful `ACK(0)` packet to the accessory. The accessory should not assume that any tracks are playing, but should use `GetPlayStatus` to verify that playback has started.

Note: Commands in other lingo codes may change the Apple device's playback status, including its player state, track position, and track index. Examples include the Simple Remote lingo command `ContextButtonStatus` and the Display Remote lingo command `SetCurrentPlayingTrack`. Accessories should request notifications (via `SetPlayStatusChangeNotification`) to coordinate itself with the Apple device's playback status.

Playback Status Notifications

If requested, an Apple device can generate playback status notifications and send them to the accessory whenever its playback status changes. See the `SetPlayStatusChangeNotification` and `PlayStatusChangeNotification` commands for details about setting up and receiving notifications.

IMPORTANT: Accessories that display track position updates to the user must use play status notifications. Continuous polling, using `GetPlayStatus`, is not acceptable under self-certification testing.

Notifications are sent only when status changes occur, not at regular intervals. Accessories must accept the latest `PlayStatusChangeNotification` or `ReturnPlayStatus` command they receive as representing the current Apple device state, regardless of how long ago the command was sent. The accessory must never assume that the Apple device has stopped playing if no `PlayStatusChangeNotification` command has been received recently.

As a result of actions made by other accessories, an accessory may receive a `PlayStatusChangeNotification` command indicating an unrequested change in either the Track Index or Playback Engine Contents. In such cases, the accessory must query the Apple device again for information about the currently playing track and must not simply re-assert its previous track selection.

Note: Status notification settings persist across changes in Apple device power states.

The Database Engine

The Database Engine is always accessible when the unit is awake. It can be manipulated remotely and allows groups of content items to be selected, independently of the Playback Engine. This allows the user to listen to an existing track or playlist while checking the Apple device database for another selection. Once a different database selection is made, the user selection (the track or content playlist) is sent to the Playback Engine. The commands such as `ResetDBSelection` and `GetNumberCategorizedDBRecords` are examples of commands that are used to manipulate the Database Engine. You can identify database commands from playback commands by the string "DB" or "Database" embedded in the command name.

Database Category Hierarchies

The database engine uses **categories** to classify music, videos and other records stored in the database. Possible categories are playlist, genre, media kind, artist, album, track, composer, audiobook, and podcast. A list of records can be assembled, based on the various selected categories, to create a user list of records (a playlist).

The database categories have a hierarchy by which records are sorted and retrieved. This category hierarchy has an impact on the order in which records must be selected. For example, if a low category, such as album, is selected first, followed by a higher relative category such as genre, the album selection is invalidated and is ignored. When creating a new set of database selections, the accessory must begin by resetting all database selections, using the `ResetDBSelection` or `ResetDBSelectionHierarchy` commands, and selecting the desired database categories from highest to lowest relative category. The category hierarchy, from highest to lowest (excluding podcasts), is shown in [Table 4-6](#) (page 419).

Table 4-6 Database category hierarchy (excluding podcasts)

Category	Notes
All (highest level)	This is the state after a <code>ResetDBSelection</code> or <code>ResetDBSelectionHierarchy</code> command. No database categories are selected. If the <code>GetNumberCategorizedDBRecords</code> command is sent while in this state, it returns the total number of records for the requested category.
Playlist	When the <code>SelectDBRecord</code> command selects a playlist, all lower database category selections (genre or media kind, artist or composer, album, and track) are invalidated.
Genre or Media Kind	When the <code>SelectDBRecord</code> command selects a genre, all lower database category selections (artist or composer, album, and track) are invalidated.
Artist or Composer	When the <code>SelectDBRecord</code> command selects an artist or composer, all album and track category selections are invalidated.
Album	When the <code>SelectDBRecord</code> command selects an album, all track category selections are invalidated.
Track or Audiobook (lowest level)	When the <code>SelectDBRecord</code> command selects a track (music or video) or an audiobook, it is automatically transferred from the Database Engine to the Playback Engine. Selecting an audiobook in the Audiobook category does not affect the selection of a track in another category, and vice versa.

There are parallel hierarchies for podcasts and audiobook tracks. When the podcast category is selected, all track or audiobook selections are invalidated. If other categories (such as Artist) are then selected after the podcast category, the list of podcasts returned by `GetNumberCategorizedDBRecords` and `RetrieveCategorizedDBRecords` is affected. This behavior parallels that of playlists with multiple categories selected. The audiobook category always lists all audiobook tracks.

The default sort order for podcasts is alphabetical. For podcast episode tracks, the order is reverse chronological; that is, the most recent one is first.

Table 4-7 Database category hierarchy for podcasts

Category	Notes
All (highest level)	This is the state after a <code>ResetDBSelection</code> or <code>ResetDBSelectionHierarchy</code> command. No database categories are selected. If the <code>GetNumberCategorizedDBRecords</code> command is sent while in this state, it returns the total number of records for the requested category.
Podcast	When the <code>SelectDBRecord</code> command selects a podcast, all lower database category selections are invalidated.
Episode Track (lowest level)	When used in the context of podcasts, episode is synonymous with track. When the <code>SelectDBRecord</code> command selects an episode track, it is automatically transferred from the Database Engine to the Playback Engine.

Database Category Counts

The record count for a given database category is established either when the `GetNumberCategorizedDBRecords` command, or when the `SelectDBRecord` command is processed for that category. Depending on the previous database state and the order in which the categories are selected, the category record counts returned by the `GetNumberCategorizedDBRecords` command may be different.

After all database selections have been reset using the `ResetDBSelection` command, sending the `GetNumberCategorizedDBRecords` command for any category retrieves the total number of records matching that category in the entire Apple device database. In contrast, if one or more categories are already selected, the `GetNumberCategorizedDBRecords` command returns the number of records that match the both the specified category, as well as any already selected categories. As an example, assume an Apple device has the following four tracks in its database:

Genre	Artist	Album	Track
Electronica/Dance	Dirty Vegas	Essential Mix	Days Go By
R&B	Radiance	Pick 'n Choose	All Night
Hip-Hop/Rap	Coolio	It Takes A Thief	Fantastic Voyage
Electronica/Dance	New Order	Power, Corruption & Lies	Blue Monday

Imagine the following command sequence:

1. The database is reset using `ResetDBSelection`. All record counts are cleared.
2. The track record count returned by `GetNumberCategorizedDBRecords` is four, the total number in the database.
3. The genre Electronica/Dance is selected. The track record count returned by `GetNumberCategorizedDBRecords` is two. Two records match the Electronica/Dance genre category: the tracks by Dirty Vegas and New Order.

4. The artist Dirty Vegas is selected. Because the genre category is still selected, the track record count returned by `GetNumberCategorizedDBRecords` is one. One record matches both the Electronica/Dance genre category and the Dirty Vegas artist category: the track "Days Go By."

Note that in this example, each time a category is selected the track count is invalidated because it is in a lower category than the category selected. See ["Database Category Hierarchies"](#) (page 419) for details about the effects a category selection has on other category selections.

Retrieving the record count of a higher category does not cause the current selection to be invalidated. However, selecting one of the results of that count does cause an invalidation. For example, using the same Apple device database, a different command sequence produces different results:

1. The database is reset using `ResetDBSelection`.
2. The artist Dirty Vegas is selected. The Track record count returned by `GetNumberCategorizedDBRecords` for the Artist category is one, as there is only one track matching the Dirty Vegas artist category: the track "Days Go By."
3. The Genre record count is requested using `GetNumberCategorizedDBRecords`; this also returns a record count of one. Had the database included multiple Dirty Vegas songs from the same genre, then the returned Genre record count would still have been one. Had the database included Dirty Vegas songs from multiple different genres, however, the returned Genre record count would have been greater than one.
4. The Electronica/Dance genre is selected, which results in the Dirty Vegas selection being invalidated. If the `GetNumberCategorizedDBRecords` command is used to obtain the record count for the Artist category, it returns two, the expected total number of artists matching the Electronica/Dance genre (Dirty Vegas and New Order).

The All Tracks Playlist

The Apple device's All Tracks playlist contains every track in the current hierarchy (either audio or video) that is stored on the Apple device and always resides at index 0x00 in the Playlist category.

Accessories can use the following steps to select and play all the tracks on the Apple device:

1. Call `ResetDBSelection()`.
2. Call `GetNumberCategorizedDatabaseRecords(playlists)`. This returns the number of playlists available on the Apple device.
3. Call `SelectDBRecord(playlists, 0)`. This selects the All Tracks playlist.
4. Call `PlayCurrentSelection()`.

Nested Playlists

Version 1.13 and later of the Extended Interface protocol supports nested playlists. A playlist can contain either tracks or one or more other playlists, but not both. The number of playlist nesting levels is unlimited.

When navigating playlists using the Apple device's Playlist category, nested playlists are flattened and their contents are presented in the highest level playlist that contains them. For example, if playlist Adam contains two nested playlists, Bob and Carol, then the Playlist category displays three playlists: Adam, Bob, and Carol. The contents of playlists Bob and Carol are their tracks. The contents of playlist Adam is the concatenation of the tracks in playlists Bob and Carol.

The Apple device provides a Nested Playlist category to traverse playlists with their nesting hierarchy in place. In the foregoing example, only playlist Adam would be returned when traversing the Nested Playlist category after a `ResetDBSelection` command. Selecting playlist Adam would expose nested playlists Bob and Carol. Selecting Bob or Carol would expose no further nested playlists. A typical command sequence for traversing the hierarchy looks like this:

1. Call `ResetDBSelection`; all record counts are cleared.
2. Call `GetNumberCategorizedDBRecords`; a `NestedPlaylist` count of 1 is returned.
3. Call `SelectDBRecord`; select a `NestedPlaylist` index of 0 (playlist Adam).
4. Call `GetNumberCategorizedDBRecords`; a `NestedPlaylist` count of 2 is returned.
5. Call `SelectDBRecord`; select a `NestedPlaylist` index of 1 (playlist Carol).
6. Call `GetNumberCategorizedDBRecords`; a `NestedPlaylist` count of 0 is returned.

When the count of nested playlists returned by `GetNumberCategorizedDBRecords` is 0, the current playlist has no nested playlists. At this point the accessory should query the Apple device for the number and names of the tracks in that playlist.

Calling `SelectDBRecord` with a `NestedPlaylist` index of -1 traverses the hierarchy upward to the next parent playlist. If the current playlist has no parent playlist, passing index -1 has no effect.

Unexpected Database Changes

Wireless iTunes store access, rentals, Genius playlists, and other media features present opportunities for the Apple device's database contents to change without synchronizing to the iTunes database on a host computer. Category counts, category lists, track counts, tracks, and other media information may change at any time, including during Extended Interface sessions. Playback engine counts and contents may also change as media are added to or removed from the Apple device.

Accessories must not expect the media information and content to remain the same throughout an Extended Interface session. Accessories must verify media information immediately before use to ensure that the data has not changed, especially when media information is cached locally on the accessory. Even then, the database or playback engine state may change without the Apple device informing the accessory. Accessories using the Extended Interface must be programmed to recover from command errors and failures received unexpectedly from the Apple device, by retrying commands or by reidentifying themselves and entering Extended Interface mode again.

Transferring Album Art

The extended interface lingo includes several commands that support the transfer of album artwork from an Apple device to an accessory:

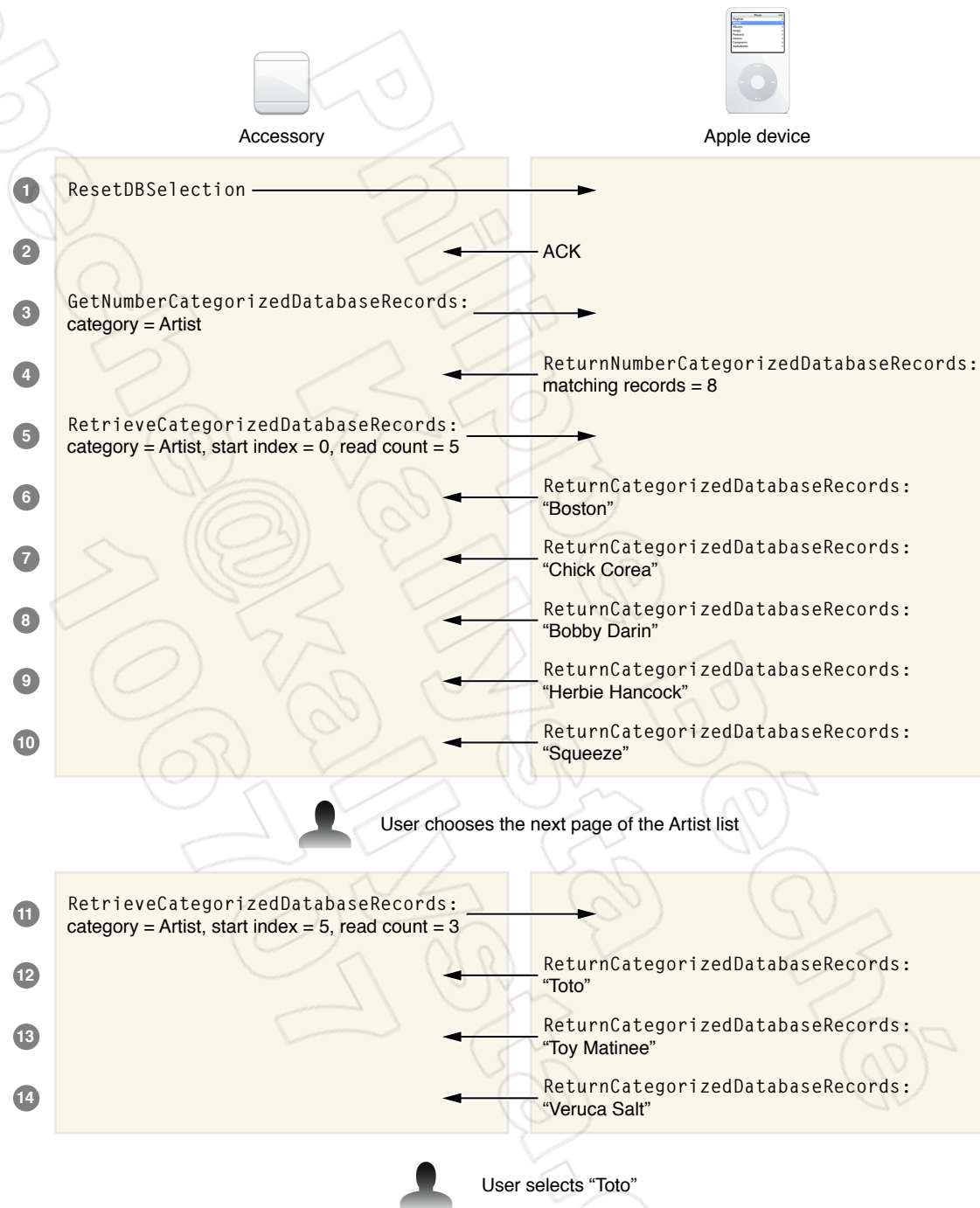
- `GetArtworkFormats`
- `RetArtworkFormats`
- `GetTrackArtworkTimes`
- `RetTrackArtworkTimes`
- `GetTrackArtworkData`
- `RetTrackArtworkData`

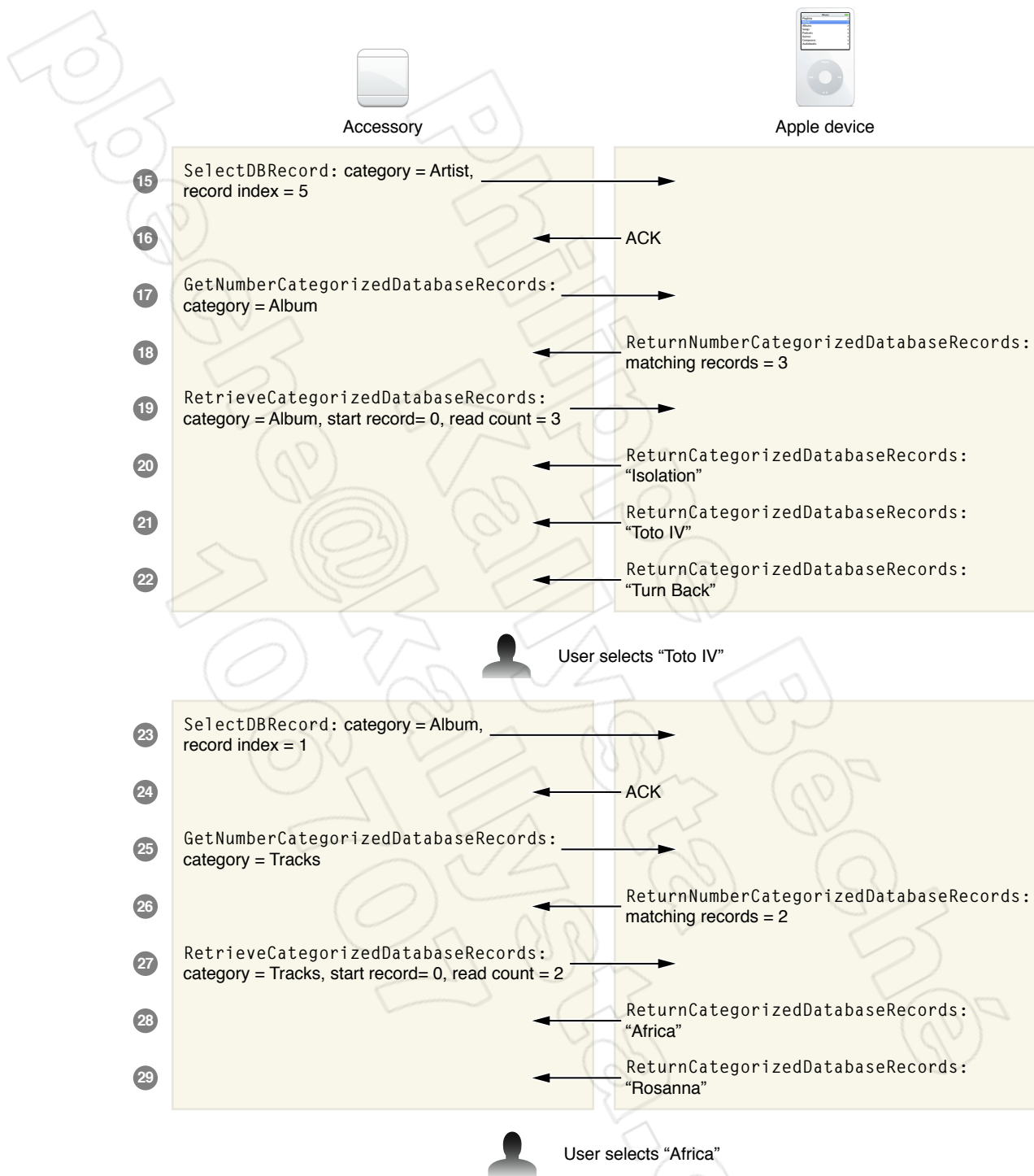
All album art image encoding is currently RGB-565, which can be transferred in both big- and little-endian formats. Artwork retrieval takes place in the following steps:

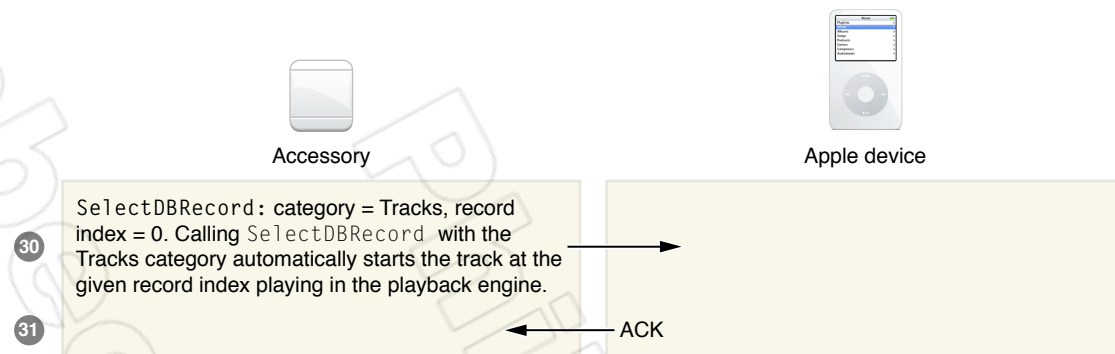
1. Retrieve the number of formats available for artwork on the Apple device using `GetArtworkFormats`. It is not necessary to call `GetArtworkFormats` more than once per session; these values will be static while the accessory is attached to the Apple device. However, there are no guarantees about the number of formats and which ones are available on a particular model or firmware version. Each `formatID` in `RetArtworkFormats` specifies both a pixel encoding, such as RGB-565 little-endian, and the image dimensions. All formats are fixed-size.
2. When the accessory wants to retrieve the artwork for a given track, it calls `GetIndexedPlayingTrackInfo` with an `infoType` of 7. This returns the count of artwork available for each `formatID` associated with the track. It is possible that a track may not have artwork for a particular `formatID` or that the number of images will vary by `formatID`. A given size of album artwork may be available for only one track in the database.
3. To retrieve the list of images associated with a given track and `formatID`, the accessory calls `GetTrackArtworkTimes`. This command tells the Apple device to return the associated timestamp for each artwork. The timestamp indicates when the artwork should be displayed, expressed in milliseconds from the start of playback.
4. When the accessory wants to retrieve an individual piece of artwork, it sends `GetTrackArtworkData` to the Apple device. This requires the accessory to specify a track, a `formatID`, and the timestamp of the desired image. The Apple device returns the specified artwork and the accessory can display it whenever it chooses.

Using the Extended Interface Protocol: An Example

The following example of command traffic shows a conversation that might occur if the interfacing accessory has a paged menu of 5 choices; that is, a menu capable of displaying up to 5 options for the user to choose from.

Figure 4-2 An example interaction between an Extended Interface accessory and an Apple device





Video Browsing

With Apple devices that store and display video content (in addition to music), an accessory may choose to navigate the Apple device's hierarchy of stored videos. To do this, the accessory must use the command `ResetDBSelectionHierarchy` to switch from navigating music tracks to navigating video tracks.

Video Playlists

A video playlist is a collection of tracks, created in iTunes, that contains one or more tracks playable as video. In contrast, an audio-only playlist does not contain any tracks playable as video. Video playlists may contain video-only tracks (such as movies) or tracks playable as either video or audio (such as video podcasts or music videos). Audio-only tracks are not visible and not playable when the Apple device is navigating the video hierarchy.

Video Navigation User Interface

Video track selection follows the Apple device user interface rules as closely as possible and behaves in the same way as audio track selection. In the video hierarchy, the menu item `Genre` is used to indicate media kind. The list of media kinds is dynamic and may be updated in the future to add, modify, or remove entries.

Once a media kind has been selected, the existing `Artist`, `Album`, and `Song or Track` categories may be used to further narrow the selection. For some media kinds, such as movies, a category may contain a single entry. In such cases, the accessory may choose to bypass this menu level. For example, with TV shows the `Season` menu on the Apple device is omitted if all the stored episodes are from the same season. Accessory designs that provide a video browsing user interface are encouraged to duplicate this behavior.

Video podcasts and music videos appear in both the music and video hierarchies. Other video tracks, such as movies and TV shows, appear only in the video hierarchy. Note that video playlists may contain audio-only tracks.

The `Artist` and `Album` categories do not apply to every media kind. For example, movies do not use these categories at the current time. When a category does not apply, the Apple device returns a count of 1 and displays a localized version of the word "All." The accessory may choose to bypass this menu level when displaying it to the user, as described above.

Navigating Video Playlists

The `ResetDBSelectionHierarchy` command with the video hierarchy type (0x02) selects the video hierarchy and invalidates all previous video database selections, such as `ResetDBSelection` in the audio-only hierarchy. As with the audio hierarchy, a video playlist record index of 0x00 designates the all video tracks playlist. If there are no video tracks on an Apple device, the all video tracks playlist will still be present, with a record count of 0.

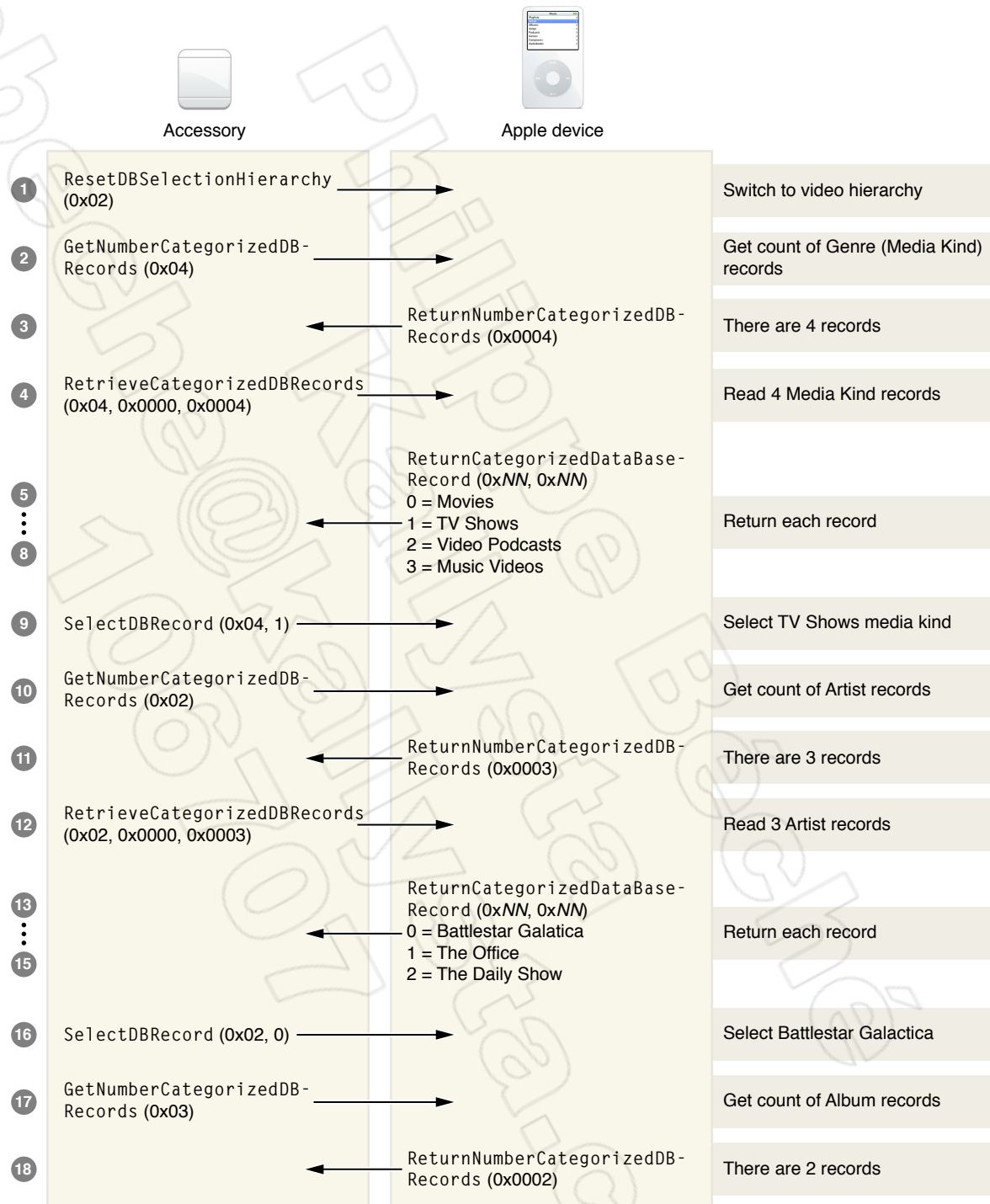
After a `ResetDBSelectionHierarchy` command is sent to select the video hierarchy, all video playlists become visible. The video playlist count can be obtained using the `GetNumberCategorizedDBRecords` command with the playlist category (0x01). The video playlist records can be obtained using the `RetrieveCategorizedDatabaseRecords` command. A specific video playlist can be selected using the `SelectDBRecord` command.

Compatibility Note: 5G Apple devices with firmware version 1.2.1 and earlier (before 11/2007) require that 1 be added to the `recIndex` values for music video artists. These Apple devices also do not filter out audio-only tracks while navigating the video hierarchy.

Video Selection Examples

[Figure 4-3](#) (page 428) diagrams a typical interchange that might take place between an accessory and an Apple device as the user of the accessory navigates to a TV show stored on an Apple device.

TV shows and video podcasts use both the Artist and Album categories. Music videos return “All” as the only Album entry. Movies return “All” for both Artist and Album categories. However, this behavior may change in future versions of the Apple device firmware. Accessories must treat these categories as dynamic.

Figure 4-3 Navigating to a TV show: example of interactions between an accessory and an Apple device

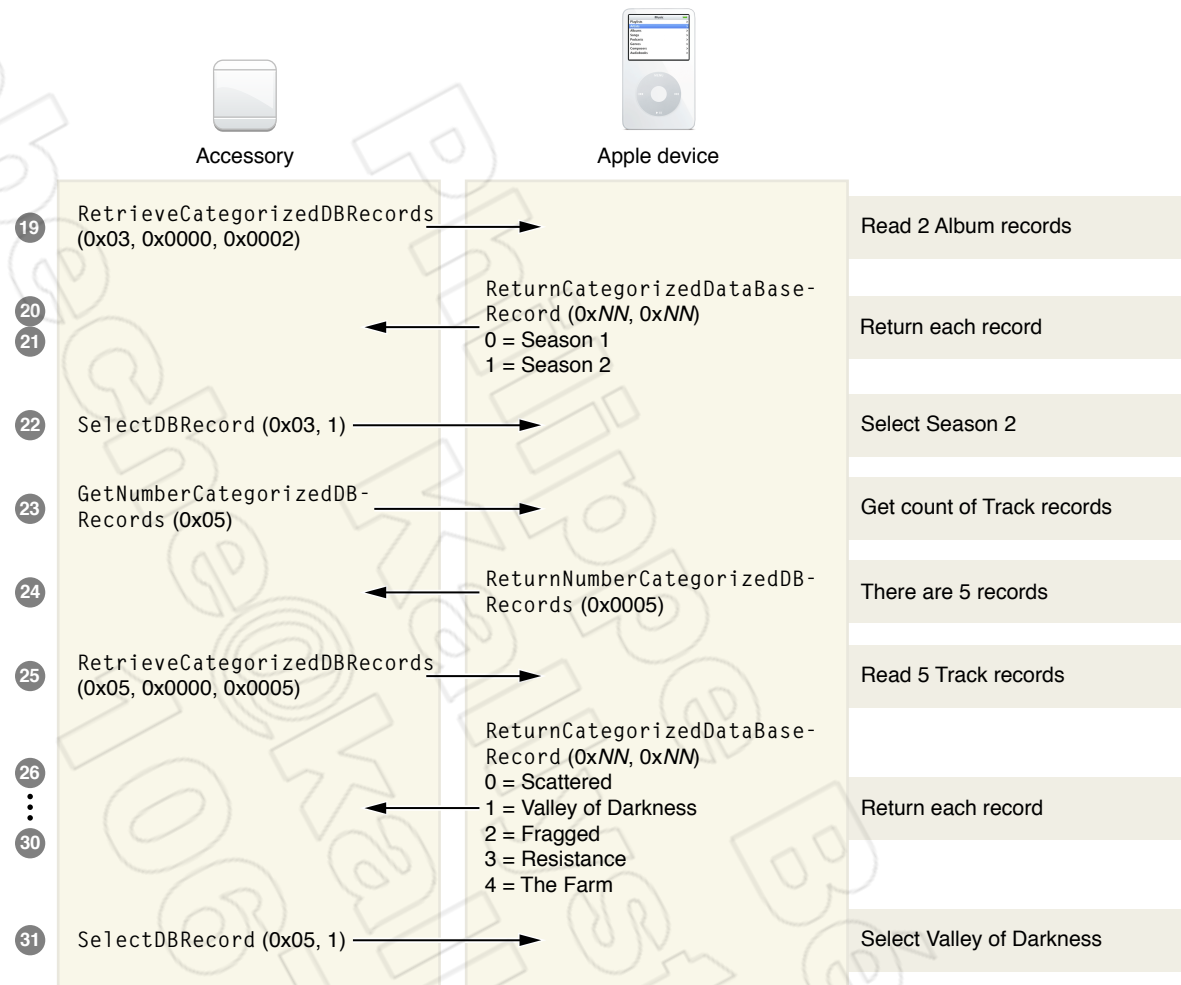
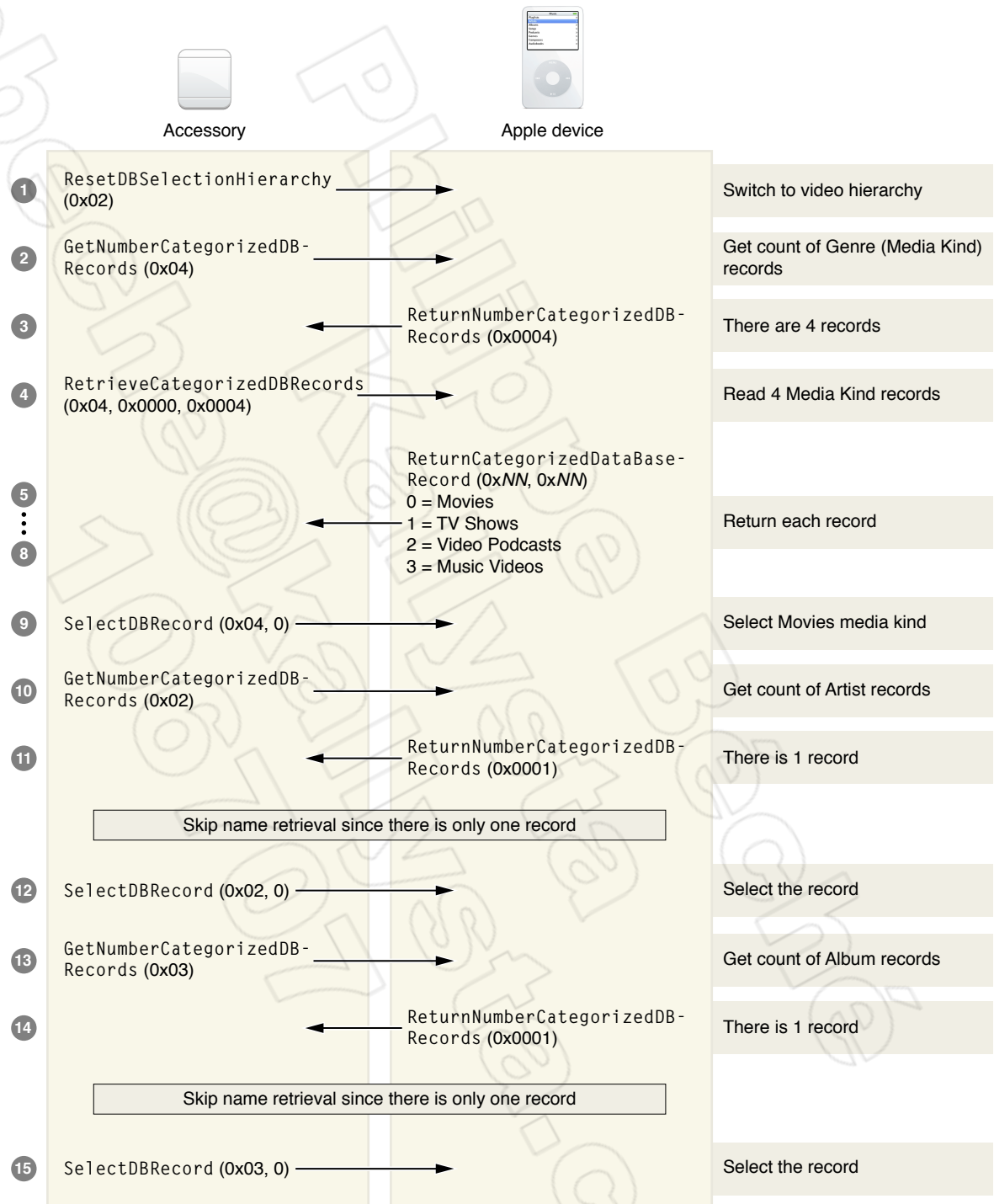
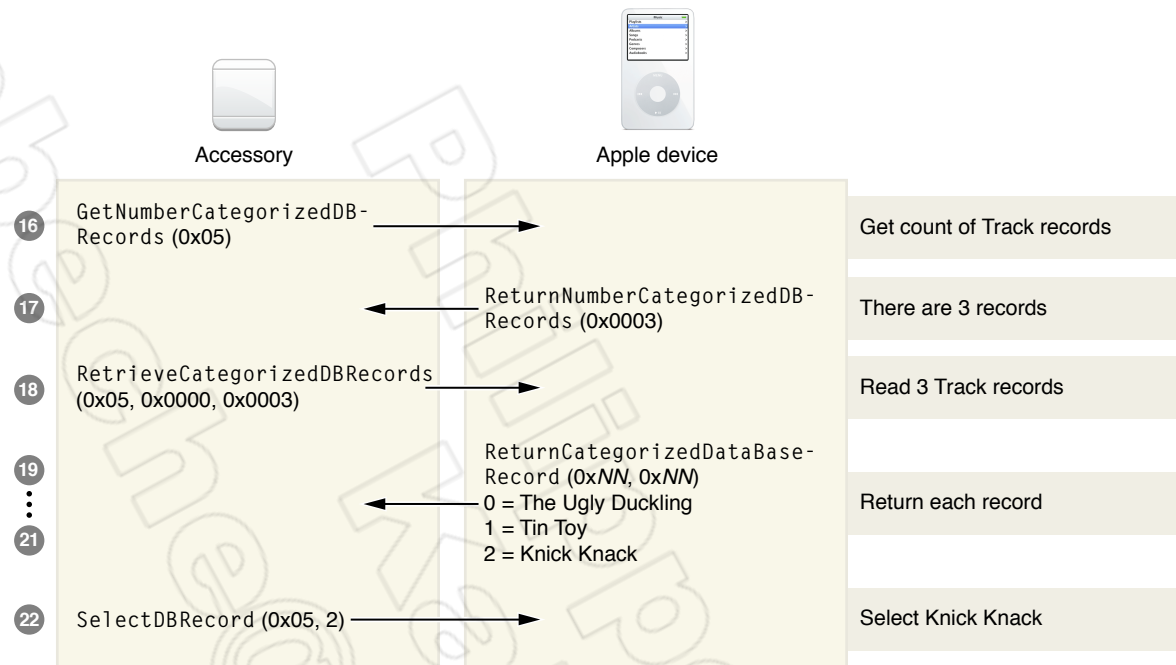


Figure 4-4 (page 430) diagrams the same kind of process with a movie. Note that in the case of the movie, the hierarchy search drops through two levels because those levels each contain only one record.

Figure 4-4 Navigating to a movie: example of interactions between an accessory and an Apple device



In video browsing, categorized DB records can be retrieved in any order, following essentially the same rules as navigating audio categories. It is not necessary to select the media kind. [Table 4-8](#) (page 431) illustrates the process of selecting a playlist that contains video material.

Table 4-8 Selecting playlists containing video

Step	Action or command	Direction	Comments
1	Pass 0x02 (video) in <code>ResetDBSelection-Hierarchy</code>	Acc to Dev	Switch to video hierarchy.
2	Pass 0x01 (playlist) in <code>GetNumberCategorizedDBRecords</code>	Acc to Dev	Get count of playlists containing video. Any playlist that has at least one video track will be counted.
3	Receive 0x05 from <code>ReturnNumber-CategorizedDBRecords</code>	Dev to Acc	There are 5 records.
4	Pass (0x01, 0x0000, 0x0005) in <code>RetrieveCategorizedDatabaseRecords</code>	Acc to Dev	Get 5 video playlists starting with index 0.
5–9	Receive data from <code>ReturnCategorized-DataBaseRecord</code>	Dev to Acc	Return 5 record strings from the database.
10	Pass (0x01, 0x0001) in <code>SelectDBRecord</code>	Acc to Dev	Select video playlist index 1. Only video tracks will be displayed.

Following the steps shown above, the video track selection process can follow steps 16 to 22 shown in [Navigating to a movie: example of interactions between an accessory and an Apple device](#) (page 431).

Sample Extended Interface Session

Table 4-9 (page 432) shows a typical sequence of Extended Interface commands. This command sequence explores the Apple device's database and then returns information about a specific track.

Table 4-9 Typical extended interface commands

Step	Accessory command	Apple device command	Comment
The accessory identifies itself, is authenticated by Apple device, and enters extended interface mode.			
1	GetDBiTunesInfo(0x00)		Accessory requests database UID from Apple device. The UID is used by the accessory to determine if this Apple device database has been attached before.
2		RetDBiTunesInfo	The Apple device returns database UID. The accessory checks the UID to see if it has a match to previously cached DB contents.
3	GetDBiTunesInfo(0x01)		The accessory requests database last sync date/time.
4		RetDBiTunesInfo	The Apple device returns database last sync date/time. If DB has been cached by the accessory, the date/time is checked to see if the DB has been synced since the last time.
5	GetDBiTunesInfo(0x02)		The accessory requests database total audio track count.
6		RetDBiTunesInfo	The Apple device returns database total audio track count. If DB has been cached by the accessory, the audio track count is checked to see if the DB has changed since the contents were cached. A count of 0 (no tracks are present) can be used to gray out or remove audio options in the accessory's user interface.
7	GetDBiTunesInfo(0x03)		The accessory requests database total video track count.

Step	Accessory command	Apple device command	Comment
8		RetDBiTunesInfo	The Apple device returns database total video track count. If DB has been cached by the accessory, the video track count is checked to see if the DB has changed since the contents were cached. A count of 0 (no tracks are present) can be used to gray out or remove video options in the accessory's user interface.
9	GetDBiTunesInfo(0x04)		The accessory requests database total audiobook track count.
10		RetDBiTunesInfo	The Apple device returns database total audiobook track count. If DB has been cached by the accessory, the audiobook track count is checked to see if the DB has changed since the contents were cached.
11	ResetDBSelection		The accessory resets database selection before getting the total track count.
12		ACK	The Apple device acknowledges the ResetDBSelection command.
13	GetNumberCategorized-DBRecords(0x05)		The accessory gets the database track count.
14		RetNumberCategorized-DBRecords	The Apple device returns the database track count.
15	GetDBTrackInfo(0, trackCount, infoMask)		The accessory requests track information for all audio tracks. The infoMask tells the Apple device what types of information should be returned for each track.
16		RetDBTrackInfo(N, infoType, trackInfo)	The Apple device returns track index N information of type infoType.
...			
17		RetDBTrackInfo(N, infoType+t, trackInfo)	The Apple device returns track index N information of type infoType+t.
18		RetDBTrackInfo(N+1, infoType, trackInfo)	The Apple device returns track index N+1 information of type infoType.

Step	Accessory command	Apple device command	Comment
...			
19		RetDBTrackInfo(N+1, infoType+t, trackInfo)	The Apple device returns track index N+1 information of type infoType+t.
...			
20		RetDBTrackInfo(N+n, infoType, trackInfo)	The Apple device returns track index N+n information of type infoType.
...			
21		RetDBTrackInfo(N+n, infoType+t, trackInfo)	The Apple device returns track index N+n information of type infoType+t.
22	GetNumPlayingTracks		The accessory gets the playing track count.
23		RetNumPlayingTracks	The Apple device returns the playing track count.
24	GetPBTrackInfo(0, trackCount, infoMask)		The accessory requests track information for all audio tracks. The infoMask tells Apple device what types of information should be returned for each track.
25		RetPBTrackInfo(N, infoType, trackInfo)	The Apple device returns track index N information of type infoType.
...			
26		RetPBTrackInfo(N, infoType+t, trackInfo)	The Apple device returns track index N information of type infoType+t.
27		RetPBTrackInfo(N+1, infoType, trackInfo)	The Apple device returns track index N+1 information of type infoType.
...			
28		RetPBTrackInfo(N+1, infoType+t, trackInfo)	The Apple device returns track index N+1 information of type infoType+t.
...			
29		RetPBTrackInfo(N+n, infoType, trackInfo)	The Apple device returns track index N+n information of type infoType.
...			

Step	Accessory command	Apple device command	Comment
30		RetPBTrackInfo(N+n, infoType+t, trackInfo)	The Apple device returns track index N+n information of type infoType+t.
31	GetUIDTrackInfo(trackUID, infoMask)		The accessory requests track information for the specified track UID (the track UID was obtained via the GetDBTrackInfo or GetPBTrackInfo commands). The infoMask tells the Apple device what types of information should be returned for the track.
32		RetUIDTrackInfo(UID, infoType, trackInfo)	The Apple device returns track UID information of type infoType.
...			
33		RetUIDTrackInfo(UID, infoType+t, trackInfo)	The Apple device returns track UID information of type infoType+t.

Command Packets

This section describes the individual Extended Interface (Lingo 0x04) command packets.

Unless otherwise specified, the following rules apply:

- All packet data fields larger than 8 bits are sent and received in big-endian format; that is, ordered from the most significant byte to the least significant byte.
- Accessory command packets that have a valid checksum but contain an invalid parameter, invalid command, or other such failure cause the Apple device to respond with an ACK command containing the appropriate error status.
- A packet with an invalid checksum received by Apple device is presumed to be invalid and is ignored. No ACK or other command is sent to the accessory in response to the invalid packet.
- All UTF-8 strings returned from the Apple device are null-terminated.
- All commands require authentication before they can be used over the USB port link. Some commands require authentication when used over a serial link; see [Table 4-10](#) (page 436) for a list.

Command Code Summary

[Table 4-10](#) (page 436) lists the valid command codes for the Extended Interface protocol.

Table 4-10 Extended Interface lingo command summary

Command ID	Command	Target engine	Introduced in protocol version	Requires authentication over serial link
0x0000	Reserved	N/A	N/A	N/A
0x0001	ACK	N/A	1.00	No
0x0002	GetCurrentPlaying-TrackChapterInfo	Playback Engine	1.06	No
0x0003	ReturnCurrentPlaying-TrackChapterInfo	Playback Engine	1.06	No
0x0004	SetCurrentPlaying-TrackChapter	Playback Engine	1.06	No
0x0005	GetCurrentPlaying-TrackChapterPlayStatus	Playback Engine	1.06	No
0x0006	ReturnCurrentPlaying-TrackChapterPlayStatus	Playback Engine	1.06	No
0x0007	GetCurrentPlaying-TrackChapterName	Playback Engine	1.06	No
0x0008	ReturnCurrentPlaying-TrackChapterName	Playback Engine	1.06	No
0x0009	GetAudiobookSpeed	N/A	1.06	No
0x000A	ReturnAudiobookSpeed	N/A	1.06	No
0x000B	SetAudiobookSpeed	N/A	1.06	No
0x000C	GetIndexedPlaying-TrackInfo	Playback Engine	1.08	No
0x000D	ReturnIndexedPlaying-TrackInfo	N/A	1.08	No
0x000E	GetArtworkFormats	Playback Engine	1.10	No
0x000F	RetArtworkFormats	Playback Engine	1.10	No
0x0010	GetTrackArtworkData	Playback Engine	1.10	No
0x0011	RetTrackArtworkData	Playback Engine	1.10	No
0x0012	RequestProtocolVersion	See "Deprecated Extended Interface Commands" (page 615).		

Command ID	Command	Target engine	Introduced in protocol version	Requires authentication over serial link
0x0013	ReturnProtocolVersion	See "Deprecated Extended Interface Commands" (page 615).		
0x0014	RequestiPodName	See "Deprecated Extended Interface Commands" (page 615).		
0x0015	ReturniPodName	See "Deprecated Extended Interface Commands" (page 615).		
0x0016	ResetDBSelection	Database Engine	1.00	No
0x0017	SelectDBRecord	Database Engine As an optimization, selecting a single track or audiobook automatically passes it to the player.	1.00	No
0x0018	GetNumberCategorized-DBRecords	Database Engine	1.00	No
0x0019	ReturnNumber-CategorizedDBRecords	N/A	1.00	No
0x001A	RetrieveCategorized-DatabaseRecords	Database Engine	1.00	No
0x001B	ReturnCategorized-DatabaseRecord	N/A	1.00	No
0x001C	GetPlayStatus	Playback Engine	1.00	No
0x001D	ReturnPlayStatus	N/A	1.00	No
0x001E	GetCurrentPlaying-TrackIndex	Playback Engine	1.00	No
0x001F	ReturnCurrentPlaying-TrackIndex	N/A	1.00	No
0x0020	GetIndexedPlaying-TrackTitle	Playback Engine	1.00	No
0x0021	ReturnIndexedPlaying-TrackTitle	N/A	1.00	No
0x0022	GetIndexedPlaying-TrackArtistName	Playback Engine	1.00	No

Command ID	Command	Target engine	Introduced in protocol version	Requires authentication over serial link
0x0023	ReturnIndexedPlaying-TrackArtistName	N/A	1.00	No
0x0024	GetIndexedPlaying-TrackAlbumName	Playback Engine	1.00	No
0x0025	ReturnIndexedPlaying-TrackAlbumName	N/A	1.00	No
0x0026	SetPlayStatusChange-Notification	Playback Engine	1.00	No
0x0027	PlayStatusChange-Notification	N/A	1.00	No
0x0028	PlayCurrentSelection	Database and Playback Engines. This command copies items from the database to the Playback Engine.	1.00	No
0x0029	PlayControl	Playback Engine	1.00	No
0x002A	GetTrackArtworkTimes	Playback Engine	1.10	No
0x002B	RetTrackArtworkTimes	Playback Engine	1.10	No
0x002C	GetShuffle	N/A	1.00	No
0x002D	ReturnShuffle	N/A	1.00	No
0x002E	SetShuffle	N/A	1.00	No
0x002F	GetRepeat	N/A	1.00	No
0x0030	ReturnRepeat	N/A	1.00	No
0x0031	SetRepeat	N/A	1.00	No
0x0032	SetDisplayImage	N/A	1.01	No
0x0033	GetMonoDisplay-ImageLimits	N/A	1.01	No
0x0034	ReturnMonoDisplay-ImageLimits	N/A	1.01	No
0x0035	GetNumPlayingTracks	Playback Engine	1.01	No
0x0036	ReturnNumPlayingTracks	N/A	1.01	No

Command ID	Command	Target engine	Introduced in protocol version	Requires authentication over serial link
0x0037	SetCurrentPlayingTrack	Playback Engine	1.01	No
0x0038	SelectSortDBRecord	Deprecated; do not use in new accessory designs. See "Command 0x0038: SelectSortDBRecord" (page 618).		
0x0039	GetColorDisplay-ImageLimits	N/A	1.09	No
0x003A	ReturnColorDisplay-ImageLimits	N/A	1.09	No
0x003B	ResetDBSelection-Hierarchy	Database Engine	1.11	Yes
0x003C	GetDBiTunesInfo	Database Engine	1.13	Yes
0x003D	RetDBiTunesInfo	Database Engine	1.13	Yes
0x003E	GetUIDTrackInfo	Database Engine	1.13	Yes
0x003F	RetUIDTrackInfo	Database Engine	1.13	Yes
0x0040	GetDBTrackInfo	Database Engine	1.13	Yes
0x0041	RetDBTrackInfo	Database Engine	1.13	Yes
0x0042	GetPBTrackInfo	Playback Engine	1.13	Yes
0x0043	RetPBTrackInfo	Playback Engine	1.13	Yes
0x0044	CreateGeniusPlaylist	Database Engine and Playback Engine	1.13	Yes
0x0045	RefreshGeniusPlaylist	Database Engine	1.13	Yes
0x0046	Reserved	N/A	N/A	N/A
0x0047	IsGeniusAvailable-ForTrack	Database Engine and Playback Engine	1.13	Yes
0x0048	GetPlaylistInfo	Database Engine	1.13	Yes
0x0049	RetPlaylistInfo	Database Engine	1.13	Yes
0x004A – 0xFFFF	Reserved	N/A	N/A	N/A

Command 0x0001: ACK

Direction: Apple device to Accessory

The Apple device sends this command to acknowledge the receipt of a command and return the command status. The command ID field indicates the accessory command for which the response is being sent. The command status indicates the results of the command (success or failure).

Table 4-11 ACK command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x06	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x01	Command ID (bits 7:0)
6	0xNN	Command result status. Possible values are: <ul style="list-style-type: none"> ■ 0x00 = Success (OK) ■ 0x01 = ERROR: Unknown database category ■ 0x02 = ERROR: Command failed ■ 0x03 = ERROR: Out of resources ■ 0x04 = ERROR: Bad parameter ■ 0x05 = ERROR: Unknown ID ■ 0x06 = Reserved ■ 0x07 = ERROR: Accessory not authenticated ■ 0x08 – 0x11 = Reserved ■ 0x12 = ERROR: Selection not genius ■ 0x13 – 0xFF = Reserved
7	0xNN	The ID of the command being acknowledged (bits 15:8).
8	0xNN	The ID of the command being acknowledged (bits 7:0).
9	0xNN	Packet payload checksum byte

Command 0x0002: GetCurrentPlayingTrackChapterInfo

Direction: Accessory to Apple device

Applies To: Playback Engine

Requests the chapter information of the currently playing track. In response, the Apple device sends a "Command 0x0003: ReturnCurrentPlayingTrackChapterInfo" (page 441) command to the accessory.

Note: The returned track index is valid only when there is a currently playing or paused track.

Table 4-12 GetCurrentPlayingTrackChapterInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x02	Command ID (bits 7:0)
6	0xF7	Packet payload checksum byte

Command 0x0003: ReturnCurrentPlayingTrackChapterInfo

Direction: Apple device to Accessory

Returns the chapter information of the currently playing track. The Apple device sends this command in response to the "Command 0x0002: GetCurrentPlayingTrackChapterInfo" (page 440) command from the accessory. The track chapter information includes the currently playing track's chapter index, as well as the total number of chapters in the track. The track chapter and the total number of chapters are 32-bit signed integers. The chapter index of the first chapter is always 0x00000000. If the track does not have chapter information, a chapter index of -1 (0xFFFFFFFF) and a chapter count of 0 (0x00000000) are returned.

Table 4-13 ReturnCurrentPlayingTrackChapterInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0B	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x03	Command ID (bits 7:0)

Byte number	Value	Meaning
6	0xNN	Current chapter index (bits 31:24)
7	0xNN	Current chapter index (bits 23:16)
8	0xNN	Current chapter index (bits 15:8)
9	0xNN	Current chapter index (bits 7:0)
10	0xNN	Chapter count (bits 31:24)
11	0xNN	Chapter count (bits 23:16)
12	0xNN	Chapter count (bits 15:8)
13	0xNN	Chapter count (bits 7:0)
14	0xNN	Packet payload checksum byte

Command 0x0004: SetCurrentPlayingTrackChapter

Direction: Accessory to Apple device

Applies To: Playback Engine

Sets the currently playing track chapter. You can send the ["Command 0x0002: GetCurrentPlayingTrackChapterInfo"](#) (page 440) command to get the chapter count and the index of the currently playing chapter in the current track. In response to the SetCurrentPlayingTrackChapter command, the Apple device sends an ACK command with the command status.

Note: This command should be used only when the Apple device is in a playing or paused state. The command fails if the Apple device is stopped or if the track does not contain chapter information.

Table 4-14 SetCurrentPlayingTrackChapter command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x04	Command ID (bits 7:0)
6	0xNN	Chapter index (bits 31:24)

Byte number	Value	Meaning
7	0xNN	Chapter index (bits 23:16)
8	0xNN	Chapter index (bits 15:8)
9	0xNN	Chapter index (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x0005: GetCurrentPlayingTrackChapterPlayStatus

Direction: Accessory to Apple device

Applies To: Playback Engine

Requests the chapter playtime status of the currently playing track. The status includes the chapter length and the time elapsed within that chapter. In response to a valid command, the Apple device sends a "[Command 0x0006: ReturnCurrentPlayingTrackChapterPlayStatus](#)" (page 444) command to the accessory.

Note: If the packet length or chapter index is invalid—for instance, if the track does not contain chapter information—the Apple device responds with an ACK command including the specific error status.

Table 4-15 GetCurrentPlayingTrackChapterPlayStatus command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x05	Command ID (bits 7:0)
6	0xNN	Currently playing chapter index (bits 31:24)
7	0xNN	Currently playing chapter index (bits 23:16)
8	0xNN	Currently playing chapter index (bits 15:8)
9	0xNN	Currently playing chapter index (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x0006: ReturnCurrentPlayingTrackChapterPlayStatus

Direction: Apple device to Accessory

Returns the play status of the currently playing track chapter. The Apple device sends this command in response to the "[Command 0x0005: GetCurrentPlayingTrackChapterPlayStatus](#)" (page 443) command from the accessory. The returned information includes the chapter length and elapsed time, in milliseconds.

Table 4-16 ReturnCurrentPlayingTrackChapterPlayStatus command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0B	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x06	Command ID (bits 7:0)
6	0xNN	Chapter length in milliseconds (bits 31:24)
7	0xNN	Chapter length in milliseconds (bits 23:16)
8	0xNN	Chapter length in milliseconds (bits 15:8)
9	0xNN	Chapter length in milliseconds (bits 7:0)
10	0xNN	Elapsed time in chapter, in milliseconds (bits 31:24)
11	0xNN	Elapsed time in chapter, in milliseconds (bits 23:16)
12	0xNN	Elapsed time in chapter, in milliseconds (bits 15:8)
13	0xNN	Elapsed time in chapter, in milliseconds (bits 7:0)
14	0xNN	Packet payload checksum byte

Command 0x0007: GetCurrentPlayingTrackChapterName

Direction: Accessory to Apple device

Applies To: Playback Engine

Requests a chapter name in the currently playing track. In response to a valid command, the Apple device sends a "[Command 0x0008: ReturnCurrentPlayingTrackChapterName](#)" (page 445) command to the accessory.

Note: If the received packet length or track index is invalid—for instance, if the track does not have chapter information or is not a part of the Audiobook category—the Apple device responds with an ACK command including the specific error status.

Table 4-17 GetCurrentPlayingTrackChapterName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x07	Command ID (bits 7:0)
6	0xNN	Chapter index (bits 31:24)
7	0xNN	Chapter index (bits 23:16)
8	0xNN	Chapter index (bits 15:8)
9	0xNN	Chapter index (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x0008: ReturnCurrentPlayingTrackChapterName

Direction: Apple device to Accessory

Returns a chapter name in the currently playing track. The Apple device sends this command in response to a valid "Command 0x0007: GetCurrentPlayingTrackChapterName" (page 444) command from the accessory. The chapter name is encoded as a null-terminated UTF-8 character array.

Note: The chapter name string is not limited to 252 characters; it may be sent in small or large packet format depending on the string length. The small packet format is shown.

Table 4-18 ReturnCurrentPlayingTrackChapterName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length

Byte number	Value	Meaning
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x08	Command ID (bits 7:0)
6... <i>N</i>	0x <i>NN</i>	Chapter name as UTF-8 character array
(last byte)	0x <i>NN</i>	Packet payload checksum byte

Command 0x0009: GetAudiobookSpeed

Direction: Accessory to Apple device

Requests the current Apple device audiobook speed state. The Apple device responds with the "[Command 0x000A: ReturnAudiobookSpeed](#)" (page 446) command indicating the current audiobook speed.

Table 4-19 GetAudiobookSpeed command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x09	Command ID (bits 7:0)
6	0xF0	Packet payload checksum byte

Command 0x000A: ReturnAudiobookSpeed

Direction: Apple device to Accessory

Returns the current audiobook speed setting. The Apple device sends this command in response to the "[Command 0x0009: GetAudiobookSpeed](#)" (page 446) command from the accessory.

Table 4-20 ReturnAudiobookSpeed command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0A	Command ID (bits 7:0)
6	0xNN	Audiobook speed status code. See Table 4-21 (page 447).
7	0xNN	Packet payload checksum byte

[Table 4-21](#) (page 447) shows the possible audiobook speed states returned by this command.

Table 4-21 Audiobook speed states

Value	Meaning
0xFF	Slower (–1)
0x00	Normal
0x01	Faster (+1)
0x02 – 0xFE	Reserved

Command 0x000B: SetAudiobookSpeed

Direction: Accessory to Apple device

Sets the speed of audiobook playback. The Apple device audiobook speed states are listed in [Table 4-21](#) (page 447). This command has two modes: one to set the speed of the currently playing audiobook and a second to set the audiobook speed for all audiobooks.

Byte number 7 is an optional byte; accessories must not send this byte if they want to set the speed only of the currently playing audiobook. If accessories want to set the global audiobook speed setting then they must use byte number 7. This is the Restore on Exit byte; a nonzero value restores the original audiobook speed setting when the accessory is detached. If this byte is zero, the audiobook speed setting set by the accessory is saved and persists after the accessory is detached from the Apple device. See [Table 4-23](#) (page 448) for the packet format used when including the Restore on Exit byte.

In response to this command, the Apple device sends an ACK command with the command status.

Note: Accessory developers are encouraged to always use the Restore on Exit byte with a nonzero value, to restore any of the user's Apple device settings that were modified by the accessory.

Table 4-22 SetAudiobookSpeed command to set the speed of the currently playing audiobook

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0B	Command ID (bits 7:0)
6	0xNN	New audiobook speed code. See Table 4-21 (page 447).
7	0xNN	Packet payload checksum byte

[Table 4-23](#) (page 448) shows a command to set the global audiobook speed for all audiobooks and optionally restore the setting on accessory detach.

Table 4-23 SetAudiobookSpeed command to set the global audiobook speed

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x05	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0B	Command ID (bits 7:0)
6	0xNN	Global audiobook speed code. See Table 4-21 (page 447).
7	0xNN	Restore on Exit byte. If 1, the original setting is restored on detach; if 0, the new setting persists after accessory detach.
8	0xNN	Packet payload checksum byte

Command 0x000C: GetIndexedPlayingTrackInfo

Direction: Accessory to Apple device

Applies To: Playback Engine

Gets track information for the track at the specified index. The track info type field specifies the type of information to be returned, such as current song lyrics, podcast name, episode date, and episode description. In response, the Apple device sends the "[Command 0x000D: ReturnIndexedPlayingTrackInfo](#)" (page 450) command with the requested track information. If the information type is invalid or does not apply to the selected track, the Apple device returns an ACK with an error status.

An accessory can determine the number of tracks in the list of tracks queued to play on the Apple device by sending a "[GetNumPlayingTracks](#)" (page 492) command and receiving a "[ReturnNumPlayingTracks](#)" (page 493) command in reply.

Table 4-24 GetIndexedPlayingTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0A	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0C	Command ID (bits 7:0)
6	0xNN	Track info type. See Table 4-25 (page 449).
7	0xNN	Track index (bits 31:24)
8	0xNN	Track index (bits 23:16)
9	0xNN	Track index (bits 15:8)
10	0xNN	Track index (bits 7:0)
11	0xNN	Chapter index (bits 15:8)
12	0xNN	Chapter index (bits 7:0)
(last byte)	0xNN	Packet payload checksum byte

[Table 4-25](#) (page 449) shows the types of information you can obtain for a track.

Table 4-25 Track information type

Info Type	Code
0x00	Track Capabilities and Information
0x01	Podcast Name
0x02	Track Release Date

Info Type	Code
0x03	Track Description
0x04	Track Song Lyrics
0x05	Track Genre
0x06	Track Composer
0x07	Track Artwork Count

Notes: iOS devices do not return track lyrics.

Requesting the lyrics of a track not currently being played will result in a null string being returned. Lyrics for a track being played may take up to 5 seconds to return. Use `SetPlayStatusChangeNotification` to be notified when the lyrics for the currently playing track become available.

Command 0x000D: ReturnIndexedPlayingTrackInfo

Direction: Apple device to Accessory

Returns the requested track information type and data. The Apple device sends this command in response to the "Command 0x000C: GetIndexedPlayingTrackInfo" (page 448) command. Table 4-28 (page 452) lists the available information types and their data.

Data returned as strings are encoded as null-terminated UTF-8 character arrays. If the track information string does not exist, a null UTF-8 string is returned. If the track has no release date, then the returned release date has all bytes zeros. Track song lyrics and the track description are sent in a large or small packet format with an incrementing packet index field, spanning multiple packets if needed.

Table 4-26 (page 450) shows the packet format for the `ReturnIndexedPlayingTrackInfo` command sent in response to a request for information types 0x00 to 0x02.

Table 4-26 `ReturnIndexedPlayingTrackInfo` command for info types 0x00-0x02 and 0x05-0x07

Byte number	Value	Meaning
0	0xFF	Sync byte
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0D	Command ID (bits 7:0)
6	0xNN	Track info type. See Table 4-28 (page 452).

Byte number	Value	Meaning
7 ... <i>N</i>	0x <i>NN</i>	Track information. The data format is specific to the track info type. See Table 4-28 (page 452).
(last byte)	0x <i>NN</i>	Packet payload checksum byte

[Table 4-27](#) (page 451) shows the packet format for the `ReturnIndexedPlayingTrackInfo` command sent in response to a request for information types 0x03 to 0x04. The small packet format is not shown.

Table 4-27 `ReturnIndexedPlayingTrackInfo` command for info types 0x03-0x04

Byte number	Value	Meaning
0	0xFF	Sync byte
1	0x55	Start of packet
2	0x00	Packet payload marker (large format)
3	0x <i>NN</i>	Large packet payload length (bits 15:8)
4	0x <i>NN</i>	Large packet payload length (bits 7:0)
5	0x04	Lingo ID (Extended Interface lingo)
6	0x00	Command ID (bits 15:8)
7	0x0D	Command ID (bits 7:0)
8	0x <i>NN</i>	Track info type. See Table 4-28 (page 452).
9	0x <i>NN</i>	Packet information bits. If set, these bits have the following meanings: <ul style="list-style-type: none"> ■ Bit 31:2 Reserved ■ Bit 1: This is the last packet. Applicable only if bit 0 is set. ■ Bit 0: Indicates that there are multiple packets.
10	0x <i>NN</i>	Packet Index (bits 15:8)
11	0x <i>NN</i>	Packet Index (bits 7:0)
12 ... <i>N</i>	0x <i>NN</i>	Track information as a UTF-8 string.
(last byte)	0x <i>NN</i>	Packet payload checksum byte

[Table 4-28](#) (page 452) shows the available track information types and the format of the data returned for each type.

Table 4-28 Track info types and return data

Info Type	Code	Data Format
0x00	Track Capabilities and Information	A 10-byte data. See Table 4-29 (page 452).
0x01	Podcast Name	UTF-8 string
0x02	Track Release Date	An 8-byte data. See Table 4-30 (page 453)
0x03	Track Description	UTF-8 string
0x04	Track Song Lyrics	UTF-8 string
0x05	Track Genre	UTF-8 string
0x06	Track Composer	UTF-8 string
0x07	Track Artwork Count	Artwork count data. The artwork count is a sequence of 4-byte records; each record consists of a 2-byte format ID value followed by a 2-byte count of images in that format for this track. For more information about formatID and chapter index values, see commands 0x000E-0x0011 and 0x002A-0x002B.
0x08-0xFF	Reserved	N/A

[Table 4-29](#) (page 452) shows the data returned for the Track Capabilities and Information type.

Table 4-29 Track Capabilities and Information encoding

Byte number	Code
0-3	<p>Track Capability bits. If set, these bits have the following meanings:</p> <ul style="list-style-type: none"> ■ Bits 31:15: Reserved ■ Bit 14: Track is an iTunesU episode ■ Bit 13: Track is capable of generating a Genius playlist ■ Bits 12:9: Reserved ■ Bit 8: Track is currently queued to play as a video ■ Bit 7: Track contains video (a video podcast, music video, movie, or TV show) ■ Bit 6: Track has description ■ Bit 5: Track has release date ■ Bit 4: Track is a podcast episode ■ Bit 3: Track has song lyrics ■ Bit 2: Track has album artwork ■ Bit 1: Track has chapters ■ Bit 0: Track is audiobook

Byte number	Code
4	Total track length, in milliseconds (bits 31:24)
5	Total track length, in milliseconds (bits 23:16)
6	Total track length, in milliseconds (bits 15:8)
7	Total track length, in milliseconds (bits 7:0)
8	Chapter count (bits 15:8)
9	Chapter count (bits 7:0)

Table 4-30 (page 453) shows the data returned for the Track Release Date type.

Table 4-30 Track Release Date encoding

Byte number	Code
0	Seconds (0-59)
1	Minutes (0-59)
2	Hours (0-23)
3	Day of the month (1-31)
4	Month (1-12)
5	Year (bits 15:8). For example, 2006 signifies the year 2006 AD.
6	Year (bits 7:0).
7	Weekday (0-6, where 0 = Sunday and 6 = Saturday)

Command 0x000E: GetArtworkFormats

Direction: Accessory to Apple device

Applies To: Playback Engine

The accessory sends this command to obtain the list of supported artwork formats on the Apple device. No parameters are sent. See ["Transferring Album Art"](#) (page 422).

Table 4-31 GetArtworkFormats packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Comment
2	0x03	Length of packet
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0E	Command ID (bits 7:0)
6	0xEB	Checksum

Command 0x000F: RetArtworkFormats

Direction: Apple device to Accessory

Applies To: Playback Engine

The Apple device sends this command to the accessory, giving it the list of supported artwork formats. Each format is described in a 7-byte record (formatID:2, pixelFormat:1, width:2, height:2). The `formatID` is used when sending `GetTrackArtworkTimes`. The accessory may return zero records. See ["Transferring Album Art"](#) (page 422).

Table 4-32 RetArtworkFormats packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Length of packet
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0F	Command ID (bits 7:0)
NN	0xNN	formatID (15:8) Apple device-assigned value for this format
NN	0xNN	formatID (7:0)
NN	0xNN	pixelFormat. Same as from <code>SetDisplayImage</code>
NN	0xNN	imageWidth (15:8). Number of pixels wide for each image.
NN	0xNN	imageWidth (7:0)
NN	0xNN	imageHeight (15:8). Number of pixels high for each image.
NN	0xNN	imageHeight (7:0)
Previous 7 bytes may be repeated NN times		

Byte number	Value	Comment
(last byte)	0xNN	Checksum

Command 0x0010: GetTrackArtworkData

Direction: Accessory to Apple device

Applies To: Playback Engine

The accessory sends this command to the Apple device to request data for a given `trackIndex`, `formatID`, and `artworkIndex`. See ["Transferring Album Art"](#) (page 422). The time offset from track start is the value returned by ["GetTrackArtworkTimes"](#) (page 478).

Table 4-33 GetTrackArtworkData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0D	Length of packet
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x10	Command ID (bits 7:0)
6	0xNN	trackIndex (31:24).
7	0xNN	trackIndex (23:16)
8	0xNN	trackIndex (15:8)
9	0xNN	trackIndex (7:0)
10	0xNN	formatID (15:8)
11	0xNN	formatID (7:0)
12	0xNN	time offset from track start, in ms (31:24)
13	0xNN	time offset from track start, in ms (23:16)
14	0xNN	time offset from track start, in ms (15:8)
15	0xNN	time offset from track start, in ms (7:0)
16	0xNN	Checksum

Command 0x0011: RetTrackArtworkData

Direction: Apple device to Accessory

Applies To: Playback Engine

The Apple device sends the requested artwork to the accessory. Multiple `RetTrackArtworkData` commands may be necessary to transfer all the data because it will be too much to fit into a single packet. See ["Transferring Album Art"](#) (page 422).

This command uses nearly the same format as the `SetDisplayImage` command (command 0x0032). The only difference is the addition of 2 coordinates; they define an inset rectangle that describes any padding that may have been added to the image. The coordinates consist of two x,y pairs. Each x or y value is 2 bytes, so the total size of the coordinate set is 8 bytes.

Table 4-34 `RetTrackArtworkData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Length of packet
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x11	Command ID (bits 7:0)
6	0x00	Descriptor packet index (15:8). These fields uniquely identify each packet in the <code>RetTrackArtworkData</code> transaction. The first command is the descriptor packet, which always starts with an index of 0x0000.
7	0x00	Descriptor packet index (7:0)
8	0xNN	Display pixel format code.
9	0xNN	Image width in pixels (15:8)
10	0xNN	Image width in pixels (7:0)
11	0xNN	Image height in pixels (15:8)
12	0xNN	Image height in pixels (7:0)
13	0xNN	Inset rectangle, top-left point, x value (15:8)
14	0xNN	Inset rectangle, top-left point, x value (7:0)
15	0xNN	Inset rectangle, top-left point, y value (15:8)
16	0xNN	Inset rectangle, top-left point, y value (7:0)

Byte number	Value	Comment
17	0xNN	Inset rectangle, bottom-right point, x value (15:8)
18	0xNN	Inset rectangle, bottom-right point, x value (7:0)
19	0xNN	Inset rectangle, bottom-right point, y value (15:8)
20	0xNN	Inset rectangle, bottom-right point, y value (7:0)
21	0xNN	Row size in bytes (31:24)
22	0xNN	Row size in bytes (23:16)
23	0xNN	Row size in bytes (15:8)
24	0xNN	Row size in bytes (7:0)
25-NN	0xNN...	Image pixel data (variable length)
(last byte)	0xNN	Checksum

In subsequent packets in the sequence, bytes 8 through 24 are omitted.

Command 0x0016: ResetDBSelection

Direction: Accessory to Apple device

Applies To: Database Engine

Resets the current database selection to an empty state, invalidates the category entry count (sets the count to 0) for all categories except the playlist category, and sets the database hierarchy to the audio hierarchy (even if it is currently in the video hierarchy). This is analogous to pressing a clickwheel Apple device's Menu button repeatedly to get to its top-level display. Any previously selected database items are deselected. The command has no effect on the Playback Engine. In response, the Apple device sends an ACK command with the command status.

Once the accessory has reset the database selection, it must initialize the category count before it can select database records. Please refer to "[Command 0x0018: GetNumberCategorizedDBRecords](#)" (page 460) and "[Command 0x0017: SelectDBRecord](#)" (page 458) for details.

Note: Starting with protocol version 1.07, the `ResetDBSelection` command clears the sort order.

Table 4-35 `ResetDBSelection` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x16	Command ID (bits 7:0)
6	0xE3	Packet payload checksum byte

Command 0x0017: SelectDBRecord

Direction: Accessory to Apple device

Applies To: Database Engine. Selecting a single track automatically passes it to the Playback Engine.

Selects one or more records in the Database Engine, based on a category relative index. For example, selecting category two (artist) and record index one results in a list of selected tracks (or database records) from the second artist in the artist list. [Table 4-37](#) (page 460) lists the available database categories.

Note: With iOS devices, selecting a specific track in a playlist containing only skip-when-shuffle tracks causes the entire playlist to be shuffled and passed to the Playback Engine. With other Apple devices, passing a valid track index with `SelectDBRecord` and a playlist containing only skip-when-shuffle tracks causes a single track to be passed to the Playback Engine. Use the `GetNumPlayingTracks` command to query the number of songs in the Playback Engine before using any other Playback Engine commands to alter playback.

Selections are additive and limited by the category hierarchy; see "[Database Category Hierarchies](#)" (page 419) for more information about category hierarchies. Subsequent selections are made based on the subset of records resulting from the previous selections and not from the entire database. Note that the selection of a single record automatically passes it to the Playback Engine and starts its playback. Record indices consist of a 32-bit signed integer.

`SelectDBRecord` may be called only after a category count has been initialized through a call to "[Command 0x0018: GetNumberCategorizedDBRecords](#)" (page 460). Without a valid category count, the `SelectDBRecord` call cannot select a database record and the result of calling it will be undefined. Accessories that make use of "[Command 0x0016: ResetDBSelection](#)" (page 457) must always initialize the category count before selecting a new database record using `SelectDBRecord`.

Accessories should pay close attention to the ACK returned by the `SelectDBRecord` command. Ignoring errors may cause unexpected behavior.

To undo a database selection, send the `SelectDBRecord` command with the current category selected in the Database Engine and a record index of -1 (0xFFFFFFFF). This has the same effect as pressing the iPod Menu button once and moves the database selection up to the next highest menu level. For example, if an accessory selected artist number three and then album number one, it could use the `SelectDBRecord(Album, -1)` command to return to the database selection of artist number three. If multiple database selections have been made, accessories can use any of the previously used

categories to return to the next highest database selection. If the category used in one of these `SelectDBRecord` commands has not been used in a previous database selection then the command is treated as a no-op.

Sending a `SelectDBRecord` command with the Track or Audiobook category and a record index of `-1` is invalid, because the previous database selection made with the Track category and a valid index passes the database selection to the Playback Engine. Sending a `SelectDBRecord(Track, -1)` command returns a parameter error. The Apple device also returns a bad parameter error ACK when accessories send the `SelectDBRecord` command with an invalid category type, or with the Track category and an index greater than the total number of tracks available on the Apple device.

Note: Selecting a podcast always selects from the main podcast library regardless of the current category context of the Apple device. Similarly, selecting an audiobook track selects from the list of all audiobook tracks.

To immediately go to the topmost iPod menu level and reset all database selections, send the `ResetDBSelection` command to the Apple device.

Table 4-36 `SelectDBRecord` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x08	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x17	Command ID (bits 7:0)
6	0xNN	Database category type. See Table 4-37 (page 460).
7	0xNN	Database record index (bits 31:24)
8	0xNN	Database record index (bits 23:16)
9	0xNN	Database record index (bits 15:8)
10	0xNN	Database record index (bits 7:0)
11	0xNN	Packet payload checksum byte

[Table 4-37](#) (page 460) lists the valid database categories.

Table 4-37 Database category types for commands

Category	Code	Protocol version	Comments
Top-level	0x00	1.14	<code>ReturnCategorizedDatabaseRecord</code> returns a list of top-level category names and index values. If the Apple device does not support a category, the returned name will be a null string; that category must not be used by <code>SelectDBRecord</code> commands.
Playlist	0x01	1.00	
Artist	0x02	1.00	
Album	0x03	1.00	
Genre	0x04	1.00	
Track	0x05	1.00	
Composer	0x06	1.00	
Audiobook	0x07	1.06	
Podcast	0x08	1.08	
Nested playlist	0x09	1.13	
Genius Mixes	0x0A	1.14	Similar to playlists, but with the Genius feature.
iTunesU	0x0B	1.14	Similar to podcasts
Reserved	0x0C – 0xFF	N/A	

Command 0x0018: `GetNumberCategorizedDBRecords`

Direction: Accessory to Apple device

Applies To: Database Engine

Retrieves the number of records in a particular database category. For example, an accessory can get the number of artists or albums present in the database. The category types are described in [Table 4-37](#) (page 460). The Apple device responds with a "[Command 0x0019: `ReturnNumberCategorizedDBRecords`](#)" (page 461) command indicating the number of records present for this category.

`GetNumberCategorizedDBRecords` must be called to initialize the category count before selecting a database record using "[Command 0x0017: `SelectDBRecord`](#)" (page 458). A category's record count can change based on the prior categories selected and the database hierarchy. The accessory is expected to call `GetNumberCategorizedDBRecords` in order to get the valid range of category entries before selecting a record in that category.

Note: The record count returned by this command depends on the database state before this command is sent. If the database has been reset using "[Command 0x0016: ResetDBSelection](#)" (page 457), this command returns the total number of records for a given category. However, if this command is sent after one or more categories are selected, the record count is the subset of records that are members of all the categories selected prior to this command.

Table 4-38 GetNumberCategorizedDBRecords command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x18	Command ID (bits 7:0)
6	0xNN	Database category type. See Table 4-37 (page 460).
7	0xNN	Packet payload checksum byte

Command 0x0019: ReturnNumberCategorizedDBRecords

Direction: Apple device to Accessory

Returns the number of database records matching the specified database category. The Apple device sends this command in response to the "[Command 0x0018: GetNumberCategorizedDBRecords](#)" (page 460) command from the accessory. Individual records can then be extracted by sending "[Command 0x001A: RetrieveCategorizedDatabaseRecords](#)" (page 462) to the Apple device.

If no matching database records are found, a record count of zero is returned. Category types are described in [Table 4-37](#) (page 460).

After selecting the podcast category, the number of artist, album, composer, genre, and audiobook records is always zero.

Table 4-39 ReturnNumberCategorizedDBRecords command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo

Byte number	Value	Meaning
4	0x00	Command ID (bits 15:8)
5	0x19	Command ID (bits 7:0)
6	0xNN	Database record count (bits 31:24)
7	0xNN	Database record count (bits 23:16)
8	0xNN	Database record count (bits 15:8)
9	0xNN	Database record count (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x001A: RetrieveCategorizedDatabaseRecords

Direction: Accessory to Apple device

Applies To: Database Engine

Retrieves one or more database records from the Apple device, typically based on the results from the ["Command 0x0018: GetNumberCategorizedDBRecords"](#) (page 460) query. The database category types are described in [Table 4-37](#) (page 460).

This command specifies the starting record index and the number of records to retrieve (the record count). This allows an accessory to retrieve an individual record or the entire set of records for a category. The record start index and record count consist of 32-bit signed integers. To retrieve all records from a given starting record index, set the record count to -1 (0xFFFFFFFF).

The Apple device responds to this command with a separate ["Command 0x001B: ReturnCategorizedDatabaseRecord"](#) (page 463) command for each record matching the specified criteria (category and record index range).

Table 4-40 RetrieveCategorizedDatabaseRecords command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0C	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1A	Command ID (bits 7:0)
6	0xNN	Database category type. See Table 4-37 (page 460).

Byte number	Value	Meaning
7	0xNN	Database record start index (bits 31:24)
8	0xNN	Database record start index (bits 23:16)
9	0xNN	Database record start index (bits 15:8)
10	0xNN	Database record start index (bits 7:0)
11	0xNN	Database record read count (bits 31:24)
12	0xNN	Database record read count (bits 23:16)
13	0xNN	Database record read count (bits 15:8)
14	0xNN	Database record read count (bits 7:0)
15	0xNN	Packet payload checksum byte

Command 0x001B: ReturnCategorizedDatabaseRecord

Direction: Apple device to Accessory

Contains information for a single database record. The Apple device sends one or more of these commands in response to the "Command 0x001A: RetrieveCategorizedDatabaseRecords" (page 462) command from the accessory. The category record index is included to allow the accessory to determine which record has been sent. The record data is sent as a null-terminated UTF-8 encoded data array.

Note: The database record string is not limited to 252 characters; it may be sent in small or large packet format, depending on the record size. The small packet format is shown.

Table 4-41 ReturnCategorizedDatabaseRecord command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1B	Command ID (bits 7:0)
6	0xNN	Database record category index (bits 31:24)
7	0xNN	Database record category index (bits 23:16)

Byte number	Value	Meaning
8	0xNN	Database record category index (bits 15:8)
9	0xNN	Database record category index (bits 7:0)
10...N	0xNN	Database record as a UTF-8 character array.
(last byte)	0xNN	Packet payload checksum byte

Command 0x001C: GetPlayStatus

Direction: Accessory to Apple device

Applies To: Playback Engine

Requests the current Apple device playback status, allowing the accessory to display feedback to the user. In response, the Apple device sends a ["Command 0x001D: ReturnPlayStatus"](#) (page 464) command with the current playback status.

Table 4-42 GetPlayStatus command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1C	Command ID (bits 7:0)
6	0xDD	Packet payload checksum byte

Command 0x001D: ReturnPlayStatus

Direction: Apple device to Accessory

Returns the current Apple device playback status. The Apple device sends this command in response to the ["Command 0x001C: GetPlayStatus"](#) (page 464) command from the accessory. The information returned includes the current track length, track position, and player state.

Note: The track length and track position fields are valid only if the player state is Playing or Paused. For other player states, these fields must be ignored.

Table 4-43 ReturnPlayStatus command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0C	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1D	Command ID (bits 7:0)
6	0xNN	Track length in milliseconds (bits 31:24)
7	0xNN	Track length in milliseconds (bits 23:16)
8	0xNN	Track length in milliseconds (bits 15:8)
9	0xNN	Track length in milliseconds (bits 7:0)
10	0xNN	Track position in milliseconds (bits 31:24)
11	0xNN	Track position in milliseconds (bits 23:16)
12	0xNN	Track position in milliseconds (bits 15:8)
13	0xNN	Track position in milliseconds (bits 7:0)
14	0xNN	Player state. Possible values are: <ul style="list-style-type: none"> ■ 0x00 = Stopped ■ 0x01 = Playing ■ 0x02 = Paused ■ 0x03 – 0xFE = Reserved ■ 0xFF = Error
15	0xNN	Packet payload checksum byte

Command 0x001E: GetCurrentPlayingTrackIndex

Direction: Accessory to Apple device

Applies To: Playback Engine

Requests the Playback Engine index of the currently playing track. In response, the Apple device sends a "[Command 0x001F: ReturnCurrentPlayingTrackIndex](#)" (page 466) command to the accessory.

Note: The track index returned is valid only if there is currently a track playing or paused.

Table 4-44 GetCurrentPlayingTrackIndex command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1E	Command ID (bits 7:0)
6	0xDB	Packet payload checksum byte

Command 0x001F: ReturnCurrentPlayingTrackIndex

Direction: Apple device to Accessory

Returns the Playback Engine index of the current playing track in response to the "[Command 0x001E: GetCurrentPlayingTrackIndex](#)" (page 465) command from the accessory. The track index is a 32-bit signed integer. If there is no track currently playing or paused, an index of -1 (0xFFFFFFFF) is returned.

Table 4-45 ReturnCurrentPlayingTrackIndex command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1F	Command ID (bits 7:0)
6	0xNN	Playback track index (bits 31:24)
7	0xNN	Playback track index (bits 23:16)
8	0xNN	Playback track index (bits 15:8)

Byte number	Value	Meaning
9	0xNN	Playback track index (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x0020: GetIndexedPlayingTrackTitle

Direction: Accessory to Apple device

Applies To: Playback Engine

Requests the title name of the indexed playing track from the Apple device. In response to a valid command, the Apple device sends a "Command 0x0021: ReturnIndexedPlayingTrackTitle" (page 467) command to the accessory.

Note: If the packet length or playing track index is invalid, the Apple device responds with an ACK command including the specific error status.

Table 4-46 GetIndexedPlayingTrackTitle command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x20	Command ID (bits 7:0)
6	0xNN	Playback track index (bits 31:24)
7	0xNN	Playback track index (bits 23:16)
8	0xNN	Playback track index (bits 15:8)
9	0xNN	Playback track index (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x0021: ReturnIndexedPlayingTrackTitle

Direction: Apple device to Accessory

Returns the title of the indexed playing track in response to a valid ["Command 0x0020: GetIndexedPlayingTrackTitle"](#) (page 467) command from the accessory. The track title is encoded as a null-terminated UTF-8 character array.

Note: The track title string is not limited to 252 characters; it may be sent in small or large packet format, depending on the string length. The small packet format is shown.

Table 4-47 ReturnIndexedPlayingTrackTitle command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x21	Command ID (bits 7:0)
6...N	0xNN	Track title as a UTF-8 character array. The Apple device sends a small or large command packet, depending both on the data size and the accessory's maximum packet size. Accessories must be able to handle either small or large packets containing data up to the maximum length of the packet minus packet overhead.
(last byte)	0xNN	Packet payload checksum byte

Command 0x0022: GetIndexedPlayingTrackArtistName

Direction: Accessory to Apple device

Applies To: Playback Engine

Requests the name of the artist of the indexed playing track. In response to a valid command, the Apple device sends a ["Command 0x0023: ReturnIndexedPlayingTrackArtistName"](#) (page 469) command to the accessory.

Note: If the packet length or playing track index is invalid, the Apple device responds with an ACK command including the specific error status.

Table 4-48 GetIndexedPlayingTrackArtistName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x22	Command ID (bits 7:0)
6	0xNN	Playback track index (bits 31:24)
7	0xNN	Playback track index (bits 23:16)
8	0xNN	Playback track index (bits 15:8)
9	0xNN	Playback track index (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x0023: ReturnIndexedPlayingTrackArtistName

Direction: Apple device to Accessory

Returns the artist name of the indexed playing track in response to a valid ["Command 0x0022: GetIndexedPlayingTrackArtistName"](#) (page 468) command from the accessory. The track artist name is encoded as a null-terminated UTF-8 character array.

Note: The artist name string is not limited to 252 characters; it may be sent in small or large packet format, depending on the string length. The small packet format is shown.

Table 4-49 ReturnIndexedPlayingTrackArtistName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x23	Command ID (bits 7:0)
6...N	0xNN	Artist name as UTF-8 character array
(last byte)	0xNN	Packet payload checksum byte

Command 0x0024: GetIndexedPlayingTrackAlbumName

Direction: Accessory to Apple device

Applies To: Playback Engine

Requests the album name of the indexed playing track. In response to a valid command, the Apple device sends a ["Command 0x0025: ReturnIndexedPlayingTrackAlbumName"](#) (page 470) command to the accessory.

Note: If the received packet length or playing track index is invalid, the Apple device responds with an ACK command including the specific error status.

Table 4-50 GetIndexedPlayingTrackAlbumName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x24	Command ID (bits 7:0)
6	0xNN	Playback track index (bits 31:24)
7	0xNN	Playback track index (bits 23:16)
8	0xNN	Playback track index (bits 15:8)
9	0xNN	Playback track index (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x0025: ReturnIndexedPlayingTrackAlbumName

Direction: Apple device to Accessory

Returns the album name of the indexed playing track in response to a valid ["Command 0x0024: GetIndexedPlayingTrackAlbumName"](#) (page 470) command from the accessory. The track album name is encoded as a null-terminated UTF-8 character array.

Note: The album name string is not limited to 252 characters; it may be sent in small or large packet format, depending on the string length. The small packet format is shown.

Table 4-51 ReturnIndexedPlayingTrackAlbumName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x25	Command ID (bits 7:0)
6...N	0xNN	Album name as a UTF-8 character array
(last byte)	0xNN	Packet payload checksum byte

Command 0x0026: SetPlayStatusChangeNotification

Direction: Accessory to Apple device

Applies To: Playback Engine

This command is sent by the accessory to control the status change event types sent by the Apple device.

There are two forms of the command. The one-byte form accepts a single Boolean to enable or disable all notifications for play state, track index, track time position, FFW/REW seek stop, and chapter index changes (see [Table 4-57](#) (page 474), “Play status change notification codes”). The packet for this form of the command is shown in [Table 4-52](#) (page 471). When the value of byte 6 is set to 0x00, all notifications are disabled, whether set by the one-byte or four-byte form of the command.

The four-byte form expands the status notification control into individual bits for each type of status. The packet for this form of the command is shown in [Table 4-53](#) (page 472). Bytes 6 through 9 are a fixed-length bitmask of events to be enabled or disabled, where 1 = enable, 0 = disable.

Note: The recommended way to use this command is to send the four-byte form first; if the ACK command that the Apple device sends in response passes a status other than 0x00 (Success), then send the one-byte form.

Table 4-52 One-byte SetPlayStatusChangeNotification command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Meaning
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x26	Command ID (bits 7:0)
6	0xNN	Enable/disable notifications; see Table 4-54 (page 472).
7	0xNN	Packet payload checksum byte

Table 4-53 Four-byte SetPlayStatusChangeNotification command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x26	Command ID (bits 7:0)
6	0x00	Notification event mask (bits 31:24)
7	0x00	Notification event mask (bits 23:16)
8	0xNN	Notification event mask (bits 15:8); see Table 4-55 (page 473)
9	0xNN	Notification event mask (bits 7:0); see Table 4-55 (page 473)
10	0xNN	Packet payload checksum byte

Table 4-54 One-byte status change event values

Value	Description
0x00	Disable all status event notifications
0x01	Enable play status notifications for basic play state, track index, track time position, FFW/REW seek stop, and chapter index changes (see Table 4-57 (page 474), "Play status change notification codes").
0x02-0xFF	Reserved

Table 4-55 Four-byte status change event mask bits

Bit	Description
00	Basic play state changes (stop, FFW seek stop, or REW seek stop, using status notification types 0x00, 0x02, or 0x03).
01	Extended play state changes (playback stop, FFW seek start, REW seek start, playback started, FFW/REW seek stop, or playback pause using status notification type 0x06). Uses <code>PlayControl</code> command control codes as the play status codes; see Table 4-60 (page 477).
02	Track index
03	Track time offset (ms)
04	Track time offset (sec)
05	Chapter index
06	Chapter time offset (ms)
07	Chapter time offset (sec)
08	Track unique identifier
09	Track media type (audio/video)
10	Track lyrics ready (if the track has lyrics)
11	Track capabilities changed
12	Playback engine contents changed
31:13	Reserved

Command 0x0027: PlayStatusChangeNotification

Direction: Apple device to Accessory

This command is sent by the Apple device to notify the accessory about extended interface status changes. The types of status notifications sent to the accessory are controlled by the ["SetPlayStatusChangeNotification"](#) (page 471) command.

Table 4-56 PlayStatusChangeNotification command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo

Byte number	Value	Meaning
4	0x00	Command ID (bits 15:8)
5	0x27	Command ID (bits 7:0)
6- <i>NN</i>	0x <i>NN</i>	New play status. See Table 4-57 (page 474).
(last byte)	0x <i>NN</i>	Packet payload checksum byte

Table 4-57 Play status change notification codes

Status change	Parameters	Description
Playback stopped	{0x00}	
Track index	{0x01, trackIndex; 4}	trackIndex = playback engine track index
Playback FFW seek stop	{0x02}	
Playback REW seek stop	{0x03}	
Track time offset	{0x04, trackOffsetMs; 4}	trackOffsetMs = track time offset in milliseconds
Chapter index	{0x05, chapIndex; 4}	chapIndex = chapter index
Playback status extended	{0x06, playState; 1}	playState = playback state: 0x00-0x01 = Reserved 0x02 = Stopped 0x03-0x04 = Reserved 0x05 = FFW seek started 0x06 = REW seek started 0x07 = FFW/REW seek stopped 0x08-0x09 = Reserved 0x0A = Playing 0x0B = Paused 0x0C-0xFF = Reserved
Track time offset	{0x07, trackOffsetSec; 4}	trackOffsetSec = track time offset in seconds
Chapter time offset (ms)	{0x08, chapTimeMs; 4}	chapTimeMs = chapter time offset in milliseconds
Chapter time offset (sec)	{0x09, chapTimeSec; 4}	chapTimeSec = chapter time offset in seconds
Track unique identifier	{0x0A, trackUID; 8}	trackUID = track unique identifier

Status change	Parameters	Description
Track playback mode	{0x0B, playMode; 1}	playMode = playback mode of current playing track (see Note, below): 0x00 = Audio track 0x01 = Video track 0x02-0xFF = Reserved
Track lyrics ready	{0x0C}	Lyrics for the currently playing track are available for download
Track capabilities changed	{0x0D, trackCapabilities; 4}	Track capabilities bits: Bit 31:15: Reserved Bit 14: Track is an iTunesU episode Bit 13: Track is capable of generating a Genius playlist Bit 12:9: Reserved Bit 8: Track is currently queued to play as a video Bit 7: Track contains video (a video podcast, music video, movie, or TV show) Bit 6: Track has description Bit 5: Track has release date Bit 4: Track is a podcast episode Bit 3: Track has song lyrics Bit 2: Track has album artwork Bit 1: Track has chapters Bit 0: Track is audiobook
Playback engine contents changed	{0x0E, numTracks; 4}	Number of tracks in the new playlist
Reserved	{0x0F-0xFF}	

Note: The playback mode when a track is playing may depend on the database hierarchy that was current when the track was selected. Hybrid video/audio tracks (such as music videos or video podcasts) are queued as audio tracks if selected in the audio DB hierarchy, or queued as video tracks if selected in the video DB hierarchy.

Command 0x0028: PlayCurrentSelection

Direction: Accessory to Apple device

Applies To: Playback and Database Engines. This command copies items from the database engine to the Playback Engine.

Requests playback of the currently selected track or list of tracks. The currently selected tracks are first placed in the Now Playing playlist. If the Apple device is set to shuffle, the tracks are shuffled. Then the track record index is passed to the player to play, as described in "Playback Behavior" (page 417). The Apple device also returns an ACK command indicating the status of the `PlayCurrentSelection` command.

Note: Any track that has the Skip When Shuffle attribute will be excluded from the Now Playing playlist unless its track record index is passed by this command, in which case it will play. With a playlist that contains only skip-when-shuffle tracks, the next action depends on the Apple device model. With iOS devices, the next track will be shuffled and played; with other Apple devices, no further tracks will play.

Table 4-58 `PlayCurrentSelection` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x28	Command ID (bits 7:0)
6	0xNN	Selection track record index (bits 31:24)
7	0xNN	Selection track record index (bits 23:16)
8	0xNN	Selection track record index (bits 15:8)
9	0xNN	Selection track record index (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x0029: PlayControl

Direction: Accessory to Apple device

Applies To: Playback Engine

This command is sent by the accessory to control the media playback state of the Apple device. If the Apple device is already in the requested state, the command has no effect and returns successfully. If the Apple device does not enter the requested state successfully, an error status is returned. In response, theiPod sends an ACK command with the play control status.

Note: The iPod models before the 2G nano (09/2006) always return a successful status. Starting with the 2G nano, Apple devices return the actual play control status. This means that a next or previous track command will return an error if no media is playing.

Table 4-59 PlayControl command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x29	Command ID (bits 7:0)
6	0xNN	Play control command code. See Table 4-60 (page 477).
7	0xNN	Packet payload checksum byte

[Table 4-60](#) (page 477) shows the Apple device play states the accessory can set.

Table 4-60 Play control command codes

Command	Code	Protocol	Comments
Reserved	0x00	Reserved	
Toggle Play/Pause	0x01	1.00	
Stop	0x02	1.00	
Next Track	0x03	1.00	These commands skip to the next or previous track. If the current track has played less than two seconds, Previous Track backs up to the beginning of the previous track. If the current track has played more than two seconds, it backs up to the beginning of the current track. These commands skip to the next or previous track even when the current track has chapters. To control audiobook playing by chapters, use the Next (0x08) and Previous (0x09) commands.
Previous Track	0x04	1.00	
Start FF	0x05	1.00	These commands must be followed by an End FF/Rew command (code 0x07) before any other PlayControl commands may be sent.
Start Rew	0x06	1.00	
End FF/Rew	0x07	1.00	

Command	Code	Protocol	Comments
Next	0x08	1.06	If the current track is an audiobook or a podcast with chapters, these commands skip to the next or previous chapter; see Note below. Otherwise they act like Next Track (0x03) and Previous Track (0x04).
Previous	0x09	1.06	
Play	0x0A	1.13	These commands may be used regardless of the lingo protocol version if the Apple device explicitly declares their support; see Table 2-138 (page 167).
Pause	0x0B	1.13	
Next Chapter	0x0C	1.14	Deprecated; use Next (0x08) and Previous (0x09) instead. If the current track is an audiobook or a podcast with chapters, these commands skip to the next or previous chapter; otherwise they have no effect.
Previous Chapter	0x0D	1.14	
Reserved	0x0E – 0xFF	N/A	

Note: If a track has chapters, the Next code (0x08) will advance to the beginning of the next chapter. If the track is playing its last chapter or has no chapters, the Next code will advance to the beginning of the next track. If a track has chapters and is more than two seconds into the current chapter, the Previous code (0x09) will back up to the beginning of the current chapter. If it is less than two seconds into the current chapter, the Previous code will back up to the beginning of the previous chapter. If the track has no chapters, the Previous code will back up to the beginning of the previous track.

Command: 0x002A: GetTrackArtworkTimes

Direction: Accessory to Apple device

Applies To: Playback Engine

The accessory sends this command to the Apple device to request the list of artwork time locations for a track. A 4-byte `trackIndex` specifies which track from the Playback Engine is to be selected. A 2-byte `formatID` indicates which type of artwork is desired. See ["Transferring Album Art"](#) (page 422).

The 2-byte `artworkIndex` specifies at which index to begin searching for artwork. A value of 0 indicates that the Apple device should start with the first available artwork.

The 2-byte `artworkCount` specifies the maximum number of times (artwork locations) to be returned. A value of –1 (0xFFFF) indicates that there is no preferred limit. Note that podcasts may have a large number of associated images.

Table 4-61 GetTrackArtworkTimes packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0D	Length of packet

Byte number	Value	Comment
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2A	Command ID (bits 7:0)
6	0xNN	trackIndex (31:24)
7	0xNN	trackIndex (23:16)
8	0xNN	trackIndex (15:8)
9	0xNN	trackIndex (7:0)
10	0xNN	formatID (15:8)
11	0xNN	formatID (7:0)
12	0xNN	artworkIndex (15:8)
13	0xNN	artworkIndex (7:0)
14	0xNN	artworkCount (15:8)
15	0xNN	artworkCount (7:0)
16	0xNN	Checksum

Command: 0x002B: RetTrackArtworkTimes

Direction: Apple device to Accessory

Applies To: Playback Engine

The Apple device sends this command to the accessory to return the list of artwork times for a given track. The Apple device returns zero or more 4-byte times, one for each piece of artwork associated with the track and format specified by `GetTrackArtworkTimes`. See ["Transferring Album Art"](#) (page 422).

The number of records returned will be no greater than the number specified in the `GetTrackArtworkTimes` command. It may, however, be less than requested. This can happen if there are fewer pieces of artwork available than were requested, or if the Apple device is unable to place the full number in a single packet. Check the number of records returned against the results of `ReturnIndexedPlayingTrackInfo` with `infoType 8` to ensure that all artwork has been received.

Table 4-62 RetTrackArtworkTimes packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Comment
2	0xNN	Length of packet
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2B	Command ID (bits 7:0)
6	0xNN	time offset from track start in ms (31:24)
7	0xNN	time offset from track start in ms (23:16)
8	0xNN	time offset from track start in ms (15:8)
9	0xNN	time offset from track start in ms (7:0)
Preceding 4 bytes may be repeated NN times		
(last byte)	0xNN	Checksum

Command 0x002C: GetShuffle

Direction: Accessory to Apple device

Requests the current state of the Apple device shuffle setting. The Apple device responds with the "[Command 0x002D: ReturnShuffle](#)" (page 480) command.

Table 4-63 GetShuffle command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2C	Command ID (bits 7:0)
6	0xCD	Packet payload checksum byte

Command 0x002D: ReturnShuffle

Direction: Apple device to Accessory

Returns the current state of the shuffle setting. The Apple device sends this command in response to the "Command 0x002C: GetShuffle" (page 480) command from the accessory.

Table 4-64 ReturnShuffle command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2D	Command ID (bits 7:0)
6	0xNN	Shuffle mode. See Table 4-65 (page 481).
7	0xNN	Packet payload checksum byte

[Table 4-65](#) (page 481) lists the possible values of the shuffle mode.

Table 4-65 Shuffle modes

Value	Meaning
0x00	Shuffle off
0x01	Shuffle tracks
0x02	Shuffle albums
0x03 – 0xFF	Reserved

Command 0x002E: SetShuffle

Direction: Accessory to Apple device

Sets the Apple device shuffle mode. The Apple device shuffle modes are listed in [Table 4-65](#) (page 481). In response, the Apple device sends an ACK command with the command status.

This command has an optional byte, byte 0x07, called the Restore on Exit byte. This byte can be used to restore the original shuffle setting in use when the accessory was attached to the Apple device. A nonzero value restores the original shuffle setting of the Apple device when the accessory is detached. If this byte is zero, the shuffle setting set by the accessory overwrites the original setting and persists after the accessory is detached from the Apple device.

Accessory engineers should note that the shuffle mode affects items only in the Playback Engine. The shuffle setting does not affect the order of tracks in the database engine, so calling "[Command 0x001A: RetrieveCategorizedDatabaseRecords](#)" (page 462) on a database selection with the shuffle mode set returns a list of unshuffled tracks. To get the shuffled playlist, an accessory must query the Playback Engine by calling "[Command 0x0020: GetIndexedPlayingTrackTitle](#)" (page 467).

Shuffling tracks does not affect the track index, just the track at that index. If an unshuffled track at playback index 1 is shuffled, a new track is placed into index 1. The playback indexes themselves are not shuffled.

When shuffle mode is enabled, tracks that are marked "skip when shuffling" are filtered from the database selection. This affects all tracks that the user has marked in iTunes. It also affects all podcasts unless their default "skip when shuffling" markings have been deliberately removed. To apply the filter to the Playback Engine, the accessory must send "[Command 0x0017: SelectDBRecord](#)" (page 458) or "[Command 0x0028: PlayCurrentSelection](#)" (page 475) after enabling shuffle mode.

Note: Accessory developers are encouraged to always use the Restore on Exit byte with a nonzero value to restore any settings modified by the accessory upon detach.

[Table 4-66](#) (page 482) shows a command to set the shuffle setting and optionally restore it on accessory detach.

Table 4-66 SetShuffle command with Restore on Exit byte

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x05	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2E	Command ID (bits 7:0)
6	0xNN	New shuffle mode. See Table 4-65 (page 481).
7	0xNN	Restore on Exit byte. If 1, the original setting is restored on detach; if 0, the new setting persists after accessory detach.
8	0xNN	Packet payload checksum byte

[Table 4-67](#) (page 482) shows a command to make the shuffle setting persistent after the accessory detach.

Table 4-67 SetShuffle command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2E	Command ID (bits 7:0)
6	0xNN	New shuffle mode. See Table 4-65 (page 481).
7	0xNN	Packet payload checksum byte

Command 0x002F: GetRepeat

Direction: Accessory to Apple device

Requests the track repeat state of the Apple device. In response, the Apple device sends a "[Command 0x0030: ReturnRepeat](#)" (page 483) command to the accessory.

Table 4-68 GetRepeat command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2F	Command ID (bits 7:0)
6	0xCA	Packet payload checksum byte

Command 0x0030: ReturnRepeat

Direction: iPodto Acc

Returns the current Apple device track repeat state to the accessory. The Apple device sends this command in response to the "[Command 0x002F: GetRepeat](#)" (page 483) command.

Table 4-69 ReturnRepeat command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Meaning
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x30	Command ID (bits 7:0)
6	0xNN	Repeat state. See Table 4-70 (page 484).
7	0xNN	Packet payload checksum byte

[Table 4-70](#) (page 484) lists the possible values of the repeat state field.

Table 4-70 Repeat state values

Value	Meaning
0x00	Repeat off
0x01	Repeat one track
0x02	Repeat all tracks
0x03 – 0xFF	Reserved

Command 0x0031: SetRepeat

Direction: Accessory to Apple device

Sets the repeat state of the Apple device. The Apple device track repeat modes are listed in [Table 4-70](#) (page 484). In response, the Apple device sends an ACK command with the command status.

This command has an optional byte, byte 0x07, called the Restore on Exit byte. This byte can be used to restore the original repeat setting in use when the accessory was attached to the Apple device. A nonzero value restores the original repeat setting of the Apple device when the accessory is detached. If this byte is zero, the repeat setting set by the accessory overwrites the original setting and persists after the accessory is detached from the Apple device.

Note: Accessory developers are encouraged to always use the Restore on Exit byte with a nonzero value to restore any settings modified by the accessory upon detach.

[Table 4-71](#) (page 485) shows a command to set the repeat setting and optionally restore it on accessory detach.

Table 4-71 SetRepeat command with Restore on Exit byte

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x05	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x31	Command ID (bits 7:0)
6	0xNN	New repeat state. See Table 4-70 (page 484).
7	0xNN	Restore on Exit byte. If 1, the original setting is restored on detach; if 0, the new setting persists after accessory detach.
8	0xNN	Packet payload checksum byte

[Table 4-72](#) (page 485) shows a command to make the repeat setting persistent after the accessory detach.

Table 4-72 SetRepeat command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x31	Command ID (bits 7:0)
6	0xNN	New repeat state. See Table 4-70 (page 484).
7	0xNN	Packet payload checksum byte

Command 0x0032: SetDisplayImage

Direction: Accessory to Apple device

Sets a bitmap image that is shown on the Apple device display when it is connected to the accessory. The intent is to allow third party branding when the Apple device is communicating with an external accessory. An image downloaded using this mechanism replaces the default checkmark bitmap image that is displayed when the Apple device is connected to an external accessory. The new bitmap is retained in RAM for as long

as the Apple device remains powered. After a system reset or Sleep state, the new bitmap is lost and the checkmark image restored as the default when the Apple device next enters Extended Interface mode after power-up.

Note: To determine whether the Apple device supports displaying the image sent by `SetDisplayImage`, the accessory should verify that General lingo command `RetiPodOptionsForLingo` returns bit 04 for lingo 0x04, as listed in [Table 2-138](#) (page 167).

Before setting a monochrome display image, the accessory can send the "[Command 0x0033: GetMonoDisplayImageLimits](#)" (page 491) command to obtain the current Apple device display width, height and pixel format. The monochrome display information returned in the "[Command 0x0034: ReturnMonoDisplayImageLimits](#)" (page 492) command can be useful to the accessory in deciding which type of display image format is suitable for downloading to the Apple device.

On Apple devices with color displays, accessories can send the "[Command 0x0039: GetColorDisplayImageLimits](#)" (page 494) command to obtain the Apple device color display width, height, and pixel formats. The color display information is returned in the "[Command 0x003A: ReturnColorDisplayImageLimits](#)" (page 495) command.

To set a display image, the accessory must successfully send `SetDisplayImage` descriptor and data commands to the Apple device. The `SetDisplayImage` descriptor command (packet index 0x0000) must be sent first, as it gives the Apple device a description of the image to be downloaded. This command is shown in [Table 4-73](#) (page 486)). The image descriptor command includes image pixel format, image width and height, and display row size (stride) in bytes. Optionally, the descriptor command may also contain the beginning data of the display image, as long as it remains within the maximum length limits of the packet.

Following the descriptor command, the `SetDisplayImage` data commands (packet index 0x0001 – 0xNNNN) must be sent using sequential packet indices until the entire image has been sent to the Apple device.

Note: The `SetDisplayImage` command payload length is limited to 500 bytes. This packet length limit and the size and format of the display image generally determine the minimum number of packets that are required to set a display image.

Note: Starting with the second generation iPod nano with version 1.1.2 firmware, using the `SetDisplayImage` command is limited to once every 15 seconds over USB transport. The iPod classic and iPod 3G nano apply a different restriction to both USB and UART transports: calls made to `SetDisplayImage` after the first 15 seconds will return a successful ACK command, but the bitmap will not be displayed on the Apple device's screen. Hence, use of the `SetDisplayImage` command should be limited to drawing bitmap images only immediately after entering Extended mode. The iPod touch accepts the `SetDisplayImage` command but will not draw it on its screen.

[Table 4-73](#) (page 486) shows the format of a descriptor command. This example assumes the display image descriptor data exceeds the small packet payload capacity; a large packet format is shown.

Table 4-73 `SetDisplayImage` descriptor command (packet index = 0x0000)

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Meaning
1	0x55	Start of packet
2	0x00	Packet payload marker (large format)
3	0xNN	Large packet payload length (bits 15:8)
4	0xNN	Large packet payload length (bits 7:0)
5	0x04	Lingo ID: Extended Interface lingo
6	0x00	Command ID (bits 15:8)
7	0x32	Command ID (bits 7:0)
8	0x00	Descriptor command index (bits 15:8). These fields uniquely identify each packet in the <code>SetDisplayImage</code> transaction. The first command is the Descriptor command and always starts with an index of 0x0000.
9	0x00	Descriptor command index (bits 7:0)
10	0xNN	Display pixel format code. See Table 4-75 (page 489).
11	0xNN	Image width in pixels (bits 15:8). The number of pixels, from left to right, per row.
12	0xNN	Image width in pixels (bits 7:0)
13	0xNN	Image height in pixels (bits 15:8). The number of rows, from top to bottom, in the image.
14	0xNN	Image height in pixels (bits 7:0)
15	0xNN	Row size (stride) in bytes (bits 31:24). The number of bytes representing one row of pixels. Each row is zero-padded to end on a 32-bit boundary. The cumulative size, in bytes, of the image data, transferred across all packets in this transaction is effectively $(\text{Row Size} * \text{Image Height})$.
16	0xNN	Row size (stride) in bytes (bits 23:16)
17	0xNN	Row size (stride) in bytes (bits 15:8)
18	0xNN	Row size (stride) in bytes (bits 7:0)
19 – N	0xNN	Display image pixel data
(last byte)	0xNN	Packet payload checksum byte

Note: Table 4-74 (page 488) shows the format of a data command. This example assumes the display image data exceeds the small packet payload capacity; a large packet format is shown.

Table 4-74 SetDisplayImage data packet (packet index = 0x0001 – 0xNNNN)

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x00	Packet payload marker (large format)
3	0xNN	Large packet payload length (bits 15:8)
4	0xNN	Large packet payload length (bits 7:0)
5	0x04	Lingo ID: Extended Interface lingo
6	0x00	Command ID (bits 15:8)
7	0x32	Command ID (bits 7:0)
8	0xNN	Descriptor command index (bits 15:8). These fields uniquely identify each packet in the SetDisplayImage transaction. The first command is the descriptor command, shown in Table 4-73 (page 486). The remaining n-1 commands are simply data packets, where n is determined by the size of the image.
9	0xNN	Descriptor packet index (bits 7:0)
10 – N	0xNN	Display image pixel data
(last byte)	0xNN	Packet payload checksum byte

Note: A known issue causes SetDisplayImage data packet lengths less than 11 bytes to return a bad parameter error (0x04) ACK on 3G iPods.

The Apple device display is oriented as a rectangular grid of pixels. In the horizontal direction (x-coordinate), the pixel columns are numbered, left to right, from 0 to Cmax. In the vertical direction (y-coordinate), the pixel rows are numbered, top to bottom, from 0 to Rmax. Therefore, an (x,y) coordinate of (0,0) represents the upper-leftmost pixel on the display and (Cmax,Rmax) represents the lower-rightmost pixel on the display. A portion of the Apple device display pixel layout is shown below, where x is the column number, y is the row number, and (Cmax,Rmax) represents the maximum row and column numbers.

Pixel layout	x									
y	0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0	...	Cmax,0
	0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1	...	Cmax,1

0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2	...	Cmax,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3	...	Cmax,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4	...	Cmax,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5	...	Cmax,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6	...	Cmax,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7	...	Cmax,7
...
0, Rmax	1, Rmax	2, Rmax	3, Rmax	4, Rmax	5, Rmax	6, Rmax	7, Rmax	...	Cmax,Rmax

The Apple device display pixel formats are shown in [Table 4-75](#) (page 489).

Table 4-75 Display pixel format codes

Display pixel format	Code	Protocol version
Reserved	0x00	N/A
Monochrome, 2 bits per pixel	0x01	1.01
RGB 565 color, little-endian, 16 bpp	0x02	1.09
RGB 565 color, big-endian, 16 bpp	0x03	1.09
Reserved	0x04 – 0xFF	N/A

Apple devices with color screens support all three image formats. All other Apple devices support only display pixel format 0x01 (monochrome, 2 bpp).

Display Pixel Format 0x01

Display pixel format 0x01 (monochrome, 2 bits per pixel) is the pixel format supported by all Apple devices. Each pixel consists of 2 bits that control the pixel intensity. The pixel intensities and associated binary codes are listed in [Table 4-76](#) (page 489).

Table 4-76 2 bpp monochrome pixel intensities

Pixel Intensity	Binary Code
Pixel off (not visible)	00b
Pixel on 25% (light grey)	01b
Pixel on 50% (dark grey)	10b
Pixel on 100% (black)	11b

Each byte of image data contains four packed pixels. The pixel ordering within bytes and the byte ordering within 32 bits is shown below.

Image Data Byte 0x0000							
Pixel 0		Pixel 1		Pixel 2		Pixel 3	
7	6	5	4	3	2	1	0

Image Data Byte 0x0001							
Pixel 4		Pixel 5		Pixel 6		Pixel 7	
7	6	5	4	3	2	1	0

Image Data Byte 0x0002							
Pixel 8		Pixel 9		Pixel 10		Pixel 11	
7	6	5	4	3	2	1	0

Image Data Byte 0x0003							
Pixel 12		Pixel 13		Pixel 14		Pixel 15	
7	6	5	4	3	2	1	0

Image Data Byte 0xNNNN							
Pixel N		Pixel N+1		Pixel N+2		Pixel N+3	
7	6	5	4	3	2	1	0

Display Pixel Formats 0x02 and 0x03

Display pixel format 0x02 (RGB 565, little-endian) and display pixel format 0x03 (RGB 565, big-endian) are available for use in all Apple devices with color screens. Each pixel consists of 16 bits that control the pixel intensity for the colors red, green, and blue.

It takes two bytes to represent a single pixel. Red is represented by 5 bits, green is represented by 6 bits, and blue by the final 5 bits. A 32-bit sequence represents 2 pixels. The pixel ordering within bytes and the byte ordering within 32 bits for display format 0x02 (RGB 565, little-endian) is shown below.

Image Data Byte 0x0000							
Pixel 0, lower 3 bits of green				Pixel 0, all 5 bits of blue			
7	6	5	4	3	2	1	0

Image Data Byte 0x0001							
Pixel 0, all 5 bits of red					Pixel 0, upper 3 bits of green		
7	6	5	4	3	2	1	0

Image Data Byte 0x0002							
Pixel 1, lower 3 bits of green				Pixel 1, all 5 bits of blue			
7	6	5	4	3	2	1	0

Image Data Byte 0x0003							
Pixel 1, all 5 bits of red					Pixel 1, upper 3 bits of green		
7	6	5	4	3	2	1	0

The format for display pixel format 0x03 (RGB 565, big-endian, 16 bpp) is almost identical, with the exception that bytes 0 and 1 are swapped and bytes 2 and 3 are swapped.

Command 0x0033: GetMonoDisplayImageLimits

Direction: Accessory to Apple device

Requests the limiting characteristics of the monochrome image that can be sent to the Apple device for display while it is connected to the accessory. It can be used to determine the display pixel format and maximum width and height of a monochrome image to be set using the ["Command 0x0032: SetDisplayImage"](#) (page 485) command. In response, the Apple device sends a ["Command 0x0034: ReturnMonoDisplayImageLimits"](#) (page 492) command to the accessory with the requested display information. The `GetMonoDisplayImageLimits` command is supported by Apple devices with either monochrome or color displays. To obtain color display image limits, use ["Command 0x0039: GetColorDisplayImageLimits"](#) (page 494).

Table 4-77 `GetMonoDisplayImageLimits` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x33	Command ID (bits 7:0)

Byte number	Value	Meaning
6	0xC6	Packet payload checksum byte

Command 0x0034: ReturnMonoDisplayImageLimits

Direction: Apple device to Accessory

Returns the limiting characteristics of the monochrome image that can be sent to the Apple device for display while it is connected to the accessory. The Apple device sends this command in response to the "[Command 0x0033: GetMonoDisplayImageLimits](#)" (page 491) command. Monochrome display characteristics include maximum image width and height and the display pixel format.

Table 4-78 ReturnMonoDisplayImageLimits command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x34	Command ID (bits 7:0)
6	0xNN	Maximum image width in pixels (bits 15:8)
7	0xNN	Maximum image width in pixels (bits 7:0)
8	0xNN	Maximum image height in pixels (bits 15:8)
9	0xNN	Maximum image height in pixels (bits 7:0)
10	0xNN	Display pixel format (see Table 4-75 (page 489)).
11	0xNN	Packet payload checksum byte

Command 0x0035: GetNumPlayingTracks

Direction: Accessory to Apple device

Applies To: Playback Engine

Requests the number of tracks in the list of tracks queued to play on the Apple device. In response, the Apple device sends a "[Command 0x0036: ReturnNumPlayingTracks](#)" (page 493) command with the count of tracks queued to play.

Table 4-79 GetNumPlayingTracks command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x35	Command ID (bits 7:0)
6	0xC4	Packet payload checksum byte

Command 0x0036: ReturnNumPlayingTracks

Direction: Apple device to Accessory

Returns the number of tracks in the actual list of tracks queued to play, including the currently playing track (if any). The Apple device sends this command in response to the ["Command 0x0035: GetNumPlayingTracks"](#) (page 492) command.

Table 4-80 ReturnNumPlayingTracks command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x36	Command ID (bits 7:0)
6	0xNN	Number of tracks playing (bits 31:24)
7	0xNN	Number of tracks playing (bits 23:16)
8	0xNN	Number of tracks playing (bits 15:8)
9	0xNN	Number of tracks playing (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x0037: SetCurrentPlayingTrack

Direction: Accessory to Apple device

Applies To: Playback Engine

Sets the index of the track to play in the Now Playing playlist on the Apple device. The index that is specified here is obtained by sending the "[Command 0x0035: GetNumPlayingTracks](#)" (page 492) and "[Command 0x001E: GetCurrentPlayingTrackIndex](#)" (page 465) commands to obtain the number of playing tracks and the current playing track index, respectively. In response, the Apple device sends an ACK command indicating the status of the command.

Note: This command is usable only when the Apple device is in a playing or paused state. If the Apple device is stopped, this command fails. If this command is sent with the current playing track index, the Apple device pauses playback momentarily and then resumes.

Table 4-81 SetCurrentPlayingTrack command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x37	Command ID (bits 7:0)
6	0xNN	New current playing track index (bits 31:24)
7	0xNN	New current playing track index (bits 23:16)
8	0xNN	New current playing track index (bits 15:8)
9	0xNN	New current playing track index (bits 7:0)
10	0xNN	Packet payload checksum byte

Command 0x0039: GetColorDisplayImageLimits

Direction: Accessory to Apple device

Requests the limiting characteristics of the color image that can be sent to the Apple device for display while it is connected to the accessory. It can be used to determine the display pixel format and maximum width and height of a color image to be set using the "[Command 0x0032: SetDisplayImage](#)" (page 485) command.

In response, the Apple device sends a ["Command 0x003A: ReturnColorDisplayImageLimits"](#) (page 495) command to the accessory with the requested display information. This command is supported only by Apple devices with color displays.

Table 4-82 GetColorDisplayImageLimits command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x39	Command ID (bits 7:0)
6	0xC0	Packet payload checksum byte

Command 0x003A: ReturnColorDisplayImageLimits

Direction: Apple device to Accessory

Returns the limiting characteristics of the color image that can be sent to the Apple device for display while it is connected to the accessory. The Apple device sends this command in response to the ["Command 0x0039: GetColorDisplayImageLimits"](#) (page 494) command. Display characteristics include maximum image width and height and the display pixel format.

Note: If the Apple device supports multiple display image formats, a five byte block of additional image width, height, and pixel format information is appended to the payload for each supported display format. The list of supported color display image formats returned by the Apple device may change in future software versions. Accessories must be able to parse a variable length list of supported color display formats to search for compatible formats.

Table 4-83 ReturnColorDisplayImageLimits command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3A	Command ID (bits 7:0)

Byte number	Value	Meaning
6	0xNN	Maximum image width in pixels (bits 15:8)
7	0xNN	Maximum image width in pixels (bits 7:0)
8	0xNN	Maximum image height in pixels (bits 15:8)
9	0xNN	Maximum image height in pixels (bits 7:0)
10	0xNN	Display pixel format (see Table 4-75 (page 489)).
11 - N	0xNN	Optional display image width, height, and pixel format for the second to nth supported display formats, if present.
(last byte)	0xNN	Packet payload checksum byte

Command 0x003B: ResetDBSelectionHierarchy

Direction: Accessory to Apple device

Applies To: Database Engine

This command carries a single byte in its payload (byte 6). A hierarchy selection value of 0x01 means that the accessory wants to navigate the audio hierarchy; a hierarchy selection value of 0x02 means that the accessory wants to navigate the video hierarchy.

Note: If the accessory sends a `ResetDBSelectionHierarchy` command while selecting the audio hierarchy, the database resets itself to the audio hierarchy and any video database selections are invalidated. Video selections already passed to the Playback Engine are unaffected.

Table 4-84 `ResetDBSelectionHierarchy` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3B	Command ID (bits 7:0)
6	0x01 or 0x02	Hierarchy selection
7	0xNN	Packet payload checksum byte

Note: The Apple device will return an error if an accessory attempts to enable an unsupported hierarchy, such as a video hierarchy on an Apple device that does not support video.

Command 0x003C: GetDBiTunesInfo

Direction: Accessory to Apple device

Applies to Database Engine.

This command is sent by the accessory to get the specified Apple device iTunes database metadata information. In response, the Apple device sends a `RetDBiTunesInfo` command with the requested metadata.

Note: This DB metadata information is updated when the Apple device enters remote UI mode. This ensures that the information is updated following an iTunes sync event.

Table 4-85 GetDBiTunesInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3C	Command ID (bits 7:0)
6	0xNN	iTunes database metadata type (see Table 4-86 (page 497))
7	0xNN	Packet payload checksum byte

[Table 4-86](#) (page 497) shows the iTunes database metadata type codes.

Table 4-86 iTunes database metadata types

Type code	Description
0x00	Database UID unique to an Apple device and assigned by iTunes. This ID does not change when the Apple device is synced with iTunes nor when the Apple device's content is changed. It is not the same as the track UID returned by <code>GetDBTrackInfo</code> or <code>GetPBTrackInfo</code> and used by <code>GetUIDTrackInfo</code> .
0x01	Last sync date/time to iTunes on computer; updated when the Apple device is synced with iTunes even if no content has been added or deleted.
0x02	Total audio track count (including audio podcasts and audiobooks)

Type code	Description
0x03	Total video track count (including movies, TV series, and video podcasts)
0x04	Total audiobook count
0x05	Total photo count
0x06-0xFF	Reserved

Command 0x003D: RetDBiTunesInfo

Direction: Dev to Acc

Applies to Database Engine.

This command is returned by the Apple device in response to a `GetDBiTunesInfo` command received from the accessory. If the requested iTunes metadata information type is not within the valid range, an `ACK` command with a bad parameter status is returned.

Table 4-87 RetDBiTunesInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3D	Command ID (bits 7:0)
6	0xNN	iTunes database metadata type (see Table 4-86 (page 497))
7...	...	iTunes database metadata information (see Table 4-88 (page 498))
(last byte)	0xNN	Packet payload checksum byte

[Table 4-88](#) (page 498) shows the iTunes database metadata information formats.

Table 4-88 iTunes database metadata information formats

Type code	Bytes	Description
0x00	8	iTunes database unique 64-bit identifier
0x01	7	Last sync date/time (see Table 4-89 (page 499))

Type code	Bytes	Description
0x02	4	Total audio track count
0x03	4	Total video track count
0x04	4	Total audiobook count
0x05	4	Total photo count
0x06-0xFF	Reserved	

Table 4-89 Date/time format

Byte	Description	Values
0	Seconds	00 - 59
1	Minute	00 - 59
2	Hour	00 = 12am, 23 = 11pm
3	Day	01 = 1st, 31 = 31st
4	Month	01 = Jan, 12 = Dec
5-6	Year	2007 = year 2007

Command 0x003E: GetUIDTrackInfo

Direction: Accessory to Apple device

Applies to Database Engine.

This command is sent by the accessory to get one or more types of track information using the track's Apple device-unique identifier. In response, the Apple device returns separate `RetUIDTrackInfo` commands for each type of track information requested. The track information mask is variable length; it is not necessary to transmit any trailing bytes in which no bits are set. If the track information mask contains any unrecognized track information type bits, a single `ACK` command is returned with a bad parameter status.

Note: The accessory's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the accessory in rapid succession, without any pauses or interruptions.

Table 4-90 `GetUIDTrackInfo` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3E	Command ID (bits 7:0)
6–13	0xNNNNNNNNNNNNNNNN	Unique track identifier; an accessory can obtain this value by sending a GetDBTrackInfo (page 504) or GetPBTrackInfo (page 506) command with bit 7 of byte 14 set to 1.
14...	...	Track information type bitmask (see Table 4-91 (page 500)); do not set bit 7.
(last byte)	0xNN	Packet payload checksum byte

[Table 4-91](#) (page 500) lists the track information type bitmask bits.

Table 4-91 Track information type bits

Bit	Description
0	Capabilities (media kind, skip when shuffle, has artwork, has bookmark, has lyrics, is audiobook, etc.)
1	Track name
2	Artist name
3	Album name
4	Genre name
5	Composer name
6	Total track time duration
7	Unique track identifier
8	Chapter count
9	Chapter times
10	Chapter names
11	Lyrics of the song currently playing in the Playback Engine
12	Description
13	Album track index
14	Disc set album index

Bit	Description
15	Play count
16	Skip count
17	Podcast release date
18	Last played date/time
19	Year (release date)
20	Star rating
21	Series name
22	Season number
23	Track volume adjust
24	Track EQ preset
25	Track sample rate
26	Bookmark offset
27	Start/stop time offset
28...	Reserved

Note: Requesting the lyrics of a track not currently being played will result in a null string being returned. Lyrics for a track being played may take up to 5 seconds to return. Use `SetPlayStatusChangeNotification` to be notified when the lyrics for the currently playing track become available.

Command 0x003F: RetUIDTrackInfo

Direction: Dev to Acc

Applies to Database Engine.

This command is sent by the Apple device in response to a `GetUIDTrackInfo` command received from the accessory. Multiple responses may be sent for each type of track information requested (e.g., chapter names).

Note: The accessory's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the accessory in rapid succession, without any pauses or interruptions.

Table 4-92 RetUIDTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3F	Command ID (bits 7:0)
6–13	0xNNNNNNNNNNNNNNNN	Unique track identifier
14	0xNN	Track information type (bit position number)
15...	...	Track information data (see Table 4-93 (page 502))
(last byte)	0xNN	Packet payload checksum byte

[Table 4-93](#) (page 502) shows the track information data formats.

Table 4-93 Track information data formats

Type	Description	Bytes	Format
0	Capabilities	4	See Table 4-94 (page 504)
1	Track name	NN	Null-terminated UTF8 string
2	Artist name	NN	Null-terminated UTF8 string
3	Album name	NN	Null-terminated UTF8 string
4	Genre name	NN	Null-terminated UTF8 string
5	Composer name	NN	Null-terminated UTF8 string
6	Total track duration	4	Milliseconds
7	iTunes unique track ID	8	An 8-byte UID that uniquely identifies an Apple device track. This track UID is different from the iTunes database UID returned by RetDBiTunesInfo.
8	Chapter count	2	Chapter count (0 = no chapters)

Type	Description	Bytes	Format
9	Chapter times	2,4	2 bytes chapter index (0 = first chapter) followed by 4 bytes chapter offset in milliseconds from beginning of track. A separate index/offset pair is appended for each track chapter. If a track has no chapters, no data is returned.
10	Chapter names	2,NN	2 bytes chapter index (0 = first chapter) followed by the chapter name as null-terminated UTF8 string. A separate index/name pair is appended for each track chapter. If a track has no chapters, no data is returned.
11	Lyrics of the song currently playing in the Playback Engine	2,2,NN	2 bytes current track lyrics section index (0 = first section, 0xNNNN = last section index), followed by 2 bytes maximum track lyrics section index (0 = only 1 section, 0xNNNN = maximum section index), followed by some or all of the track lyrics string. The track lyrics as a whole consists of a single null-terminated UTF8 string. If the lyrics string is too long to be carried in a single packet, then the string is broken into sections and carried in separate packets, with different values for each section index. The last lyrics packet section contains the null terminator character for the full string. Lyrics sections before the last section do not include null terminators and may not be valid UTF8 strings. Accessories must use the packet payload length to determine the length of the track lyrics string and assemble it by concatenating its substrings in order. Requesting the lyrics of a track not currently being played will result in a null string being returned.
12	Description	NN	Null-terminated UTF8 string
13	Album track index	2	index number
14	Disc set album index	2	index number
15	Play count	4	Track play count (0 = track not played)
16	Skip count	4	Track skip count (0 = track not skipped)
17	Podcast release date	7	Date/time (see Table 4-89 (page 499))
18	Last played date/time	7	Date/time (see Table 4-89 (page 499)); all zeroes if the track has never been played.
19	Year (release date)	2	Year in which track was released
20	Star rating	1	Star rating of track; 00 = No stars, 20 = 1 star, 40 = 2 stars, 60 = 3 stars, 80 = 4 stars, 100 = 5 stars.
21	Series name	NN	Null-terminated UTF8 string
22	Season number	2	Season number (1 = First season)
23	Track volume adjust	1	Track volume attenuation/amplification adjustment (0x9C = -100%, 0x00 = no adjust, 0x64 = +100%)

Type	Description	Bytes	Format
24	Track EQ preset	2	Track equalizer preset index
25	Data rate	4	Track bit rate (kilobits per second)
26	Bookmark offset	4	Bookmark offset from start of track in milliseconds
27	Start/stop time offset	4,4	4 bytes start time followed by 4 bytes stop time in milliseconds
28...	Reserved		

Table 4-94 Capabilities bits

Bit	Description
00	1 = Is audiobook
01	1 = Has chapters
02	1 = Has artwork
03	1 = Has lyrics
04	1 = Is podcast episode
05	1 = Has release date
06	1 = Has description
07	1 = Is video
08	1 = Is queued as video (in the Playback Engine only, not in the database)
12:09	Reserved
13	1 = Is capable of generating a Genius playlist
14	1 = Is an iTunesU episode
31:15	Reserved

Command 0x0040: GetDBTrackInfo

Direction: Accessory to Apple device

Applies to Database Engine.

This command is sent by the accessory to get the specified Apple device database track information types for the specified track index range. In response, the Apple device returns separate `RetDBTrackInfo` packets for each type of track information requested. The starting track index is based on the current database track selection(s). A track count of 0xFFFFFFFF (–1) returns the track information for all Apple device tracks from

the starting DB track index. The track information mask is variable length; it is not necessary to transmit any trailing bytes in which no track information bits are set. If the track index, count, or information mask contains any invalid data, a single ACK command is returned with a bad parameter status.

Note: The accessory's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the accessory in rapid succession, without any pauses or interruptions.

Table 4-95 GetDBTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x40	Command ID (bits 7:0)
6–9	0xNNNNNNNN	Track database start index
10–13	0xNNNNNNNN	Track count (from track start index)
14...	...	Track information type bitmask (see Table 4-91 (page 500))
(last byte)	0xNN	Packet payload checksum byte

Note: Requesting the lyrics of a track not currently being played will result in a null string being returned. Lyrics for a track being played may take up to 5 seconds to return. Use `SetPlayStatusChangeNotification` to be notified when the lyrics for the currently playing track become available.

Command 0x0041: RetDBTrackInfo

Direction: Dev to Acc

Applies to Database Engine.

This command is sent by the Apple device in response to a `GetDBTrackInfo` command received from the accessory. Multiple responses may be sent for each type of track information requested (e.g., chapter names).

Note: The accessory's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the accessory in rapid succession, without any pauses or interruptions.

Table 4-96 RetDBTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x41	Command ID (bits 7:0)
6–9	0xNNNNNNNN	Track database index
10	0xNN	Track information type (bit position number)
11...	...	Track information data (see Table 4-93 (page 502))
(last byte)	0xNN	Packet payload checksum byte

Command 0x0042: GetPBTrackInfo

Direction: Accessory to Apple device

Applies to Playback Engine.

This command is sent by the accessory to get the specified Apple device playing track information types for the specified track index range. In response, the Apple device returns separate RetPBTrackInfo packets for each type of track information requested. A track count of 0xFFFFFFFF (–1) returns the track information for all Apple device tracks from the starting PB track index. The track information mask is variable length; it is not necessary to transmit any trailing bytes in which no track information bits are set. If the track index, count, or information mask contains any invalid data, a single ACK command is returned with a bad parameter status.

Note: The accessory's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the accessory in rapid succession, without any pauses or interruptions.

Table 4-97 GetPBTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x42	Command ID (bits 7:0)
6–9	0xNNNNNNNN	Track playing start index
10–13	0xNNNNNNNN	Track count (from track start index)
14...	...	Track information type bitmask (see Table 4-91 (page 500))
(last byte)	0xNN	Packet payload checksum byte

Note: Requesting the lyrics of a track not currently being played will result in a null string being returned. Lyrics for a track being played may take up to 5 seconds to return. Use `SetPlayStatusChangeNotification` to be notified when the lyrics for the currently playing track become available.

Command 0x0043: RetPBTrackInfo

Direction: Dev to Acc

Applies to Playback Engine.

This command is sent by the Apple device in response to a `GetPBTrackInfo` command received from the accessory. Multiple responses may be sent for each type of track information requested (e.g., chapter names).

Note: The accessory's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the accessory in rapid succession, without any pauses or interruptions.

Table 4-98 RetPBTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x43	Command ID (bits 7:0)
6–9	0xNNNNNNNN	Track playback index
10	0xNN	Track information type (bit position number)
11...	...	Track information data (see Table 4-93 (page 502))
(last byte)	0xNN	Packet payload checksum byte

Command 0x0044: CreateGeniusPlaylist

Direction: Accessory to Apple device

Applies To: Database Engine and Playback Engine

The accessory sends this command to an Apple device to ask it to create a Genius playlist. The Apple device returns an Extended Interface ACK command, passing one of the responses listed in [Table 4-101](#) (page 509). If the playlist is successfully created, it starts playing.

Note: This command may take up to 30 seconds to finish.

Table 4-99 CreateGeniusPlaylist command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0A	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo

Byte number	Value	Meaning
4	0x00	Command ID (bits 15:8)
5	0x44	Command ID (bits 7:0): CreateGeniusPlaylist
6	0xNN	Transaction ID (bits 15:8); see "Transaction IDs" (page 525).
7	0xNN	Transaction ID (bits 7:0)
8	0xNN	Index type; see Table 4-100 (page 509).
9	0xNN	Track index (bits 31:24)
10	0xNN	Track index (bits 23:16)
11	0xNN	Track index (bits 15:8)
12	0xNN	Track index (bits 7:0)
13	0xNN	Packet payload checksum byte

Table 4-100 Index types

Value	Description
0x00	Database Engine
0x01	Playback Engine
0x02-0xFF	Reserved

Table 4-101 ACK responses to CreateGeniusPlaylist

ACK response	Meaning
0x00	Genius playlist being created. When the playlist is complete, the Apple device sends an <code>iPodNotification</code> command for Command Complete; see Table 2-132 (page 164).
0x02	Genius playlist could not be created.
0x04	Track index not valid.
0x12	Genius information not available for that track (see "Command 0x0047: IsGeniusAvailableForTrack" (page 510)).

Command 0x0045: RefreshGeniusPlaylist

Direction: Accessory to Apple device

Applies To: Database Engine

The accessory sends this command to an Apple device to ask it to refresh a Genius playlist previously created by `CreateGeniusPlaylist`. The Apple device returns an Extended Interface ACK command, passing one of the responses listed in [Table 4-103](#) (page 510).

Notes: A Genius playlist should be refreshed only after it has been selected during database browsing. This command may take up to 30 seconds to finish.

Table 4-102 RefreshGeniusPlaylist command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x09	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x45	Command ID (bits 7:0): RefreshGeniusPlaylist
6	0xNN	Transaction ID (bits 15:8); see " Transaction IDs " (page 525).
7	0xNN	Transaction ID (bits 7:0)
8	0xNN	Playlist index (bits 31:24)
9	0xNN	Playlist index (bits 23:16)
10	0xNN	Playlist index (bits 15:8)
11	0xNN	Playlist index (bits 7:0)
12	0xNN	Packet payload checksum byte

Table 4-103 ACK responses to RefreshGeniusPlaylist

ACK response	Meaning
0x00	Genius playlist being refreshed. When the refresh is complete, the Apple device sends an <code>iPodNotification</code> command for Command Complete; see Table 2-132 (page 164).
0x02	Genius playlist could not be refreshed.
0x12	Playlist is not a Genius playlist.

Command 0x0047: IsGeniusAvailableForTrack

Direction: Accessory to Apple device

Applies To: Database Engine and Playback Engine

The accessory sends this command to an Apple device to determine if Genius information is available for a given track. The Apple device returns an Extended Interface ACK command, passing one of the responses listed in [Table 4-105](#) (page 511).

Table 4-104 IsGeniusAvailableForTrack command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0A	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x47	Command ID (bits 7:0): IsGeniusAvailableForTrack
6	0xNN	Transaction ID (bits 15:8); see "Transaction IDs" (page 525).
7	0xNN	Transaction ID (bits 7:0)
8	0xNN	Index type; see Table 4-100 (page 509).
9	0xNN	Track index (bits 31:24)
10	0xNN	Track index (bits 23:16)
11	0xNN	Track index (bits 15:8)
12	0xNN	Track index (bits 7:0)
13	0xNN	Packet payload checksum byte

Table 4-105 ACK responses to IsGeniusAvailableForTrack

ACK response	Meaning
0x00	Genius information is available. This does not guarantee that the Apple device can create a Genius playlist, because doing so depends on factors such as the availability of matching songs in the database.
0x04	Track index not valid.
0x12	Track index is valid, but Genius information is not available for that track.

Command 0x0048: GetPlaylistInfo

Direction: Accessory to Apple device

Applies To: Database Engine

The accessory sends this command to an Apple device to get information about a given playlist. Currently the only information type is playlist capabilities (0x00). The Apple device returns a `RetPlaylistInfo` command, passing zero or more of the bits listed in [Table 4-107](#) (page 512).

Table 4-106 `GetPlaylistInfo` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0A	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x48	Command ID (bits 7:0): <code>GetPlaylistInfo</code>
6	0xNN	Transaction ID (bits 15:8); see "Transaction IDs" (page 525).
7	0xNN	Transaction ID (bits 7:0)
8	0xNN	Info type; see Table 4-107 (page 512).
9	0xNN	DB playlist index (bits 31:24)
10	0xNN	DB playlist index (bits 23:16)
11	0xNN	DB playlist index (bits 15:8)
12	0xNN	DB playlist index (bits 7:0)
13	0xNN	Packet payload checksum byte

Table 4-107 `GetPlaylistInfo` info types

Info type value	Description	Data length
0x00	Playlist information: Bit 0: Is Genius playlist Bits 1-31: Reserved	4 bytes
0x01-0xFF	Reserved	

Command 0x0049: `RetPlaylistInfo`

Direction: Apple device to Accessory

Applies To: Database Engine

The Apple device sends this command to an accessory in response to a `GetPlaylistInfo` command.

Table 4-108 `RetPlaylistInfo` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x49	Command ID (bits 7:0): <code>RetPlaylistInfo</code>
6	0xNN	Transaction ID (bits 15:8); see "Transaction IDs" (page 525).
7	0xNN	Transaction ID (bits 7:0)
8	0xNN	Info type; see Table 4-107 (page 512).
9-NN	0xNN	Info data; see Table 4-107 (page 512).
(last byte)	0xNN	Packet payload checksum byte

Known Issues

- Apple devices may take up to 20 seconds to detect that the remote accessory has been detached from the 30-pin connector. This depends on the firmware version and whether or not power is being supplied to the Apple device.
- External accessories such as simple remote controls may still be active, even when the Apple device has entered Extended Interface mode operation. Users should be discouraged from attaching any accessories to the Apple device while it is in Extended Interface mode.
- iPod mini version 1.4 and 4G iPod version 3.1 do not return audiobook chapter names. They return an empty string instead.

Protocol History

The following is a history of the changes made to the Extended Interface protocol.

Table 4-109 History of the Extended Interface protocol

Version	Changes
Version 1.14	Added status change control extensions to <code>SetPlayStatusChangeNotification</code> and <code>PlayStatusChangeNotification</code> .
	Added new support for Next Chapter and Previous Chapter in the <code>PlayControl</code> command.
	Added new commands 0x0044-0x0045 and 0x0047 through 0x0049.
Version 1.13	Added support for nested playlists.
	Added Play and Pause control codes to command 0x0029.
	Added new commands 0x003C through 0x0043.
Version 1.12	Improved video browsing support for <code>SelectDBRecord(-1)</code> .
Version 1.11	This version adds <code>ResetDBSelectionHierarchy</code> command to enable video browsing.
	New feature: video browsing.
	Chapter information can be retrieved for all tracks in the Now Playing list, not just for the currently playing track.
Version 1.10	This version adds commands 0x000E through 0x0011 and 0x002A through 0x002B.
	New features:
	Code restructured to improve performance.
	Added indexed playing track genre and composer info.
	Added playing track artwork support.
	Bug fixes:
Version 1.09	Return track description and lyrics info if present.
	Notify track changes even when playback is paused.
	This version adds support for color images, podcast chapters, and fixes a few bugs:
	<code>SetDisplayImage</code> supports color images on color Apple devices.
	A bug that caused <code>SelectDBRecord</code> to fail if there were no podcasts on the Apple device is fixed.
	A bug that caused <code>SelectDBRecord</code> and <code>SelectSortDBRecord</code> to reverse track order after a podcast was selected is fixed.
	<code>PlayStatusChange</code> notifications support podcast chapters.
	<code>SetCurrentPlayingChapter</code> now accepts a chapter index for podcasts.

Version	Changes
	Added new commands <code>GetColorDisplayImageLimits</code> and <code>ReturnColorDisplayImageLimits</code> to obtain information about color display images.
Version 1.08	This version adds support for the Podcast category, chapter names, and indexed playing track information. It also includes a bug fix for <code>SetDisplayImage</code> .
Version 1.07	This version adds the restore on exit feature to the set shuffle, set repeat, and set audiobook speed setting commands. It also fixes a few bugs in the database engine management commands.
	<code>ResetDBSelection</code> clears the database sort order.
	<code>SelectDBRecord</code> with an invalid index returns a command error.
	Audiobook speed restore on exit with optional flag.
	Shuffle setting restore on exit with optional flag.
	Repeat setting restore on exit with optional flag.
Version 1.06	This version adds several new lingo 0x04 commands to allow remote accessories to control audiobook playback. Please refer to lingo 0x04 commands 0x0002 through 0x000B for details. The following Extended Interface commands have been modified to support audiobook playback:
	<code>SelectDBRecord</code>
	<code>GetNumberCategorizedDBRecords</code>
	<code>ReturnNumberCategorizedDBRecords</code>
	<code>RetrieveCategorizedDatabaseRecords</code>
	<code>ReturnCategorizedDatabaseRecord</code>
	<code>PlayStatusChangeNotification</code>
	<code>PlayControl</code>
	<code>SelectSortDBRecord</code> . The sort order field is ignored for the case of an audiobook, as audiobooks are automatically sorted by name.
Version 1.05	This version adds several new lingo 0x00 commands and minor fixes to version 1.04. The changes include:
	Fixed a problem where iPod Accessory Protocol commands on the 30-pin connector did not wake up unpowered, sleeping Apple devices.
	Fixed the <code>SelectDBRecord</code> routine to play tracks from a stopped play state.
	Fixed <code>RetrieveCategorizedDBRecords</code> to retrieve all records from the start index when passed a count of -1.
	Fixed long record names resulting in large packets causing the Apple device to reboot.

Version	Changes
	Added missing begin and end FF or REW notification messages.
Version 1.04	This version adds some minor fixes to Version 1.03. The changes include:
	Fixed a problem where accessories were intermittently not detected when the 30-pin connector was plugged in to the Apple device.
	Optimized the Apple device <code>ReturnPlayStatus</code> response to the <code>GetPlayStatus</code> message from the accessory. Command response is much faster.
Version 1.03	This version is the functional equivalent of protocol version 1.02 (1.03 is equal to 1.02). It is only reported on the iPod mini in software version 1.1. All of the changes reported for version 1.02 are applicable for 1.03 and no more.
Version 1.02	This version was released in the third generation (3G) iPod with firmware version 2.2. The changes include:
	Fixed a problem where the user's On-The-Go playlist was erased when the Extended Interface serial protocol was initialized.
	Fixed a problem where a playing Apple device was stopped and playing state lost when the Extended Interface mode was initialized.
	Fixed a problem where the strings returned from the Apple device were truncated if there were multibyte characters present.
	Fixed a problem where an Apple device playing a track in Extended Interface mode would continue playing the same track, unpaused, when it was disconnected and reverted to the Apple device UI mode.
	Fixed a problem where an Apple device in Extended Interface mode failed to go into Sleep mode when power was removed from it.
	Fixed a problem where an Apple device with a set alarm would honor the alarm while in Extended Interface mode or soon after disconnection.
	Fixed a problem where the pause indicator was shown when playback was stopped during Extended Interface mode.
Version 1.01	This version was released in the iPod mini version 1.0 system software with bug fixes and protocol improvements during February 2004. The changes include:
	Added the commands: <code>SelectSortDBRecord</code> , <code>SetCurrentPlayingTrack</code> , <code>GetNumPlayingTracks</code> , <code>GetMonoDisplayImageLimits</code> and <code>SetDisplayImage</code> .
	Changed the default sort order of returned or listed database items to match that of the Apple device UI.
	Fixed a transmit problem where large packets being sent by the Apple device would pause, mid-packet, until the beginning of another packet was received by the Apple device.
	Fixed a problem where the Apple device play/pause button remained active when the Extended Interface protocol was in control of the Apple device.

Version	Changes
Version 1.0	This is the base protocol released in October 2003 in system software version 2.1 of the 3G iPod.

philip@kellys-sts.com
106707

Accessory Identification

This appendix describes the Identify Device Preferences and Settings (IDPS) process, by which an accessory may identify itself to an Apple device.

IMPORTANT: To ensure compatibility with future firmware, the designs of new accessories must use the IDPS process, except for accessories that only supply power to the Apple device. Existing accessories may continue to send the iAP General lingo command 0x13, `IdentifyDeviceLingoes`, identifying the lingoes they support and requesting immediate authentication. Note that IDPS requires accessory Authentication 2.0, as specified in "Authentication" (page 71), and the use of transaction IDs throughout each communication session, as specified in "Transaction IDs" (page 525).

Using IDPS

The IDPS process begins when an accessory sends a `StartIDPS` command to an Apple device (see "Command 0x38: `StartIDPS`" (page 128)). If the two are newly connected, it must be the first command the accessory sends. If the Apple device refuses `StartIDPS` by returning a General lingo `ACK` command with a status of 0x04 (Bad Parameter), the accessory must assume it is connected to an Apple device that doesn't support the IDPS process. In this case, the accessory must send the Apple device an `IdentifyDeviceLingoes` command within 800 ms, requesting authentication.

IMPORTANT: If the accessory has access to Accessory Power through the 30-pin connector, it must always monitor that output from the Apple device (pin 13). If Accessory Power goes low, even momentarily, the accessory must restart the IDPS process after it goes high. After Accessory Power goes high the accessory must wait at least 80 ms before starting the IDPS process or sending any other iAP commands.

During the IDPS process, the Apple device accepts only the following General lingo commands from the accessory:

- 0x4B, `GetiPodOptionsForLingo` to determine the options the Apple device supports for a specific lingo
- 0x39, `SetFIDTokenValues` to identify the accessory to the Apple device
- 0x3B, `EndIDPS` to end the IDPS process

IMPORTANT: After an Apple device has successfully acknowledged an accessory's `StartIDPS` command, all subsequent iAP command packets must include transaction IDs, regardless of lingo, as specified in "Transaction IDs" (page 525).

The IDPS process lets the Apple device receive preference information from the accessory during the initial handshake sequence. Unlike the process using `IdentifyDeviceLingoes`, the audio/video routing and other Apple device settings are established before authentication begins. Once the IDPS process begins, the Apple device will not enable preferences such as line-out and video-out by default; therefore it is up to the accessory to explicitly enable all desired preferences. During the IDPS process, the Apple device identifies connected accessories as uniquely as possible.

IMPORTANT: To avoid the Apple device displaying a compatibility warning, the accessory must start its authentication process within 2 seconds after the Apple device has detected it. Failure to do this reliably is not acceptable under self-certification testing. For the conditions under which an Apple device detects an accessory, see *MFi Accessory Hardware Specification*.

IDPS Commands

The General lingo IDPS commands that support accessory identification are listed below and documented in detail in the sections that follow. All commands are protocol version 1.09 and require authentication 2.0.

Table A-1 IDPS General lingo commands

CmdID	Name	Direction	Payload:bytes
0x38	<code>StartIDPS</code>	Acc to Dev	{transID;2}
0x39	<code>SetFIDTokenValues</code>	Acc to Dev	{transID;2, numFIDTokenValues;1, FIDTokenValues;<var>}
0x3A	<code>RetFIDTokenValueACKs</code>	Dev to Acc	{transID;2, numFIDTokenValueACKs;1, FIDTokenValueACKs;<var>}
0x3B	<code>EndIDPS</code>	Acc to Dev	{transID;2, accIDPSStatus;1}
0x3C	<code>IDPSStatus</code>	Dev to Acc	{transID;2, status;1}

Note: The `transID` parameters in the foregoing table are described in "Transaction IDs" (page 525).

For details of these commands, see "The Protocol Core and the General Lingo" (page 65).

Sample IDPS Command Sequences

This section lists two typical command sequences using IDPS:

- [Table A-2](#) (page 521) illustrates the basic IDPS command sequence.
- [Table A-3](#) (page 522) illustrates an IDPS command sequence followed by authentication and accessory support for Digital Audio (Lingo 0x0A).

Table A-2 IDPS process up to authentication

Step	Accessory command	Apple device command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
3	SetFIDTokenValues		setting 11 FID tokens ; IdentifyToken = (lingoes: 0/2/3/4 options 0x00000002 device ID 0x00000200); AccCapsToken = 0x0000000000000205; AccInfoToken = Acc name (Test-Accessory); AccInfoToken = Acc FW version (v1.1.1); AccInfoToken = Acc HW version (v2.2.2); AccInfoToken = Acc manufacturer (Apple; Inc.); AccInfoToken = Acc model number (Test-Model); EAProTOCOLToken = 1 (com.Apple.ProtocolMain); EAProTOCOLToken = 2 (com.Apple.ProtocolAlt); iPodPreferenceToken = (setting 'off' for preference class 'video out setting' with restore on exit 'not selected'); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'not selected')
4		RetFIDTokenValueACKs	11 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; EAProTOCOLToken = (1) accepted; EAProTOCOLToken = (2) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (line out usage) accepted
5	EndIDPS		status 'finished with IDPS; proceed to authentication'
6		IDPSStatus	status 'ready for auth'
7		GetDevAuthentication-Info	no params

Table A-3 IDPS process with authentication and Digital Audio lingo

Step	Accessory command	Apple device command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
3	SetFIDTokenValues		setting 11 FID tokens ; IdentifyToken = (lingoes: 0/2/3/4/10 options 0x00000002 device ID 0x00000200); AccCapsToken = 0x0000000000000215; AccInfoToken = Acc name (Test-Accessory); AccInfoToken = Acc FW version (v1.1.1); AccInfoToken = Acc HW version (v2.2.2); AccInfoToken = Acc manufacturer (Apple; Inc.); AccInfoToken = Acc model number (Test-Model); EAPProtocolToken = 1 (com.Apple.ProtocolMain); EAPProtocolToken = 2 (com.Apple.ProtocolAlt); iPodPreferenceToken = (setting 'off' for preference class 'video out setting' with restore on exit 'not selected'); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'not selected')
4		RetFIDTokenValueACKs	11 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; EAPProtocolToken = (1) accepted; EAPProtocolToken = (2) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (line out usage) accepted
5	EndIDPS		status 'finished with IDPS; proceed to authentication'
6		IDPSStatus	status 'ready for auth'
7		GetDevAuthentication-Info	no params
8	RetDevAuthentication-Info		returning version of authentication 2.0 and X.509 certificate; see Table 2-37 (page 101)

Step	Accessory command	Apple device command	Comment
9		AckDevAuthentication-Info	acknowledging 'auth info supported'
10		GetAccSampleRateCaps	no params
11		GetDevAuthentication-Signature	offering challenge for the accessory to sign and return; see Table 2-39 (page 103)
12	RetAccSampleRateCaps		returning sample rates '8000 11025 12000 16000 22050 24000 32000 44100 48000'
13	RetDevAuthentication-Signature		returning digital signature; see Table 2-40 (page 104)
14		TrackNewAudio-Attributes	sample rate 44100; Sound Check value 0; track volume adjustment 0
15	AccAck		acknowledging 'TrackNewAudioAttributes'
16		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
17		TrackNewAudio-Attributes	sample rate 44100; Sound Check value 0; track volume adjustment 0
18	AccAck		acknowledging 'TrackNewAudioAttributes'
19		TrackNewAudio-Attributes	sample rate 44100; Sound Check value 0; track volume adjustment 0
20	AccAck		acknowledging 'TrackNewAudioAttributes'

Apple Device Event Notifications

Notifications of one type, Flow Control, are enabled by default if the accessory uses IDPS. Accessories that use `IdentifyDeviceLingoes` must enable Flow Control notifications explicitly. Receiving Flow Control notifications lets accessories cope with situations where the Apple device's incoming queue is full and it is unable to accept more packets.

To work reliably with all past and future Apple devices, an accessory should first send a `GetSupportedEventNotification` command to check which notification types the Apple device supports. The Apple device can respond in one of two ways:

- The Apple device can return an ACK command with a Bad Parameter error. This means that the Apple device does not support the `GetSupportedEventNotification` command. The accessory may then send a `SetEventNotification` command, passing the `FlowControl` bit and any other bits in the returned bit mask. The Apple device may respond in one of two ways:

- ❑ If the Apple device responds with a successful `ACK` command, it means that the Apple device supports Flow Control notifications and the accessory should be prepared to handle them.
- ❑ If the Apple device responds with an `ACK` command that passes a Bad Parameter error, it means that Apple device does not support notifications.
- The Apple device can return a `RetSupportedEventNotification` command with a notification bitmask indicating which notifications it supports. The accessory may then send a `SetEventNotification` command, passing the `FlowControl` bit plus any other bits chosen from the returned bitmask.

Transaction IDs

Once an accessory has started the IDPS process, beginning with and including the `StartIDPS` command, the accessory must include 16-bit transaction ID parameters in every iAP packet. This appendix specifies the rules that must be followed when creating or interpreting the values of these parameters.

Using Transaction IDs

The purpose of transaction IDs is to uniquely associate iAP commands with their responses, regardless of the order in which commands and their responses may be transmitted.

Note: Transaction ID fields are part of the packet payload and must be counted when specifying the data lengths of iAP commands.

Generating and Returning Transaction IDs

An attached accessory's responsibility for handling transaction IDs can be summarized by the following rules:

- Every time the accessory responds to a command from an Apple device contains a transaction ID, it must include that ID in its response.
- To generate transaction IDs for the commands it sends, the accessory must initialize a 16-bit counter to 0x0000 every time it is connected to an Apple device. It must then use the counter value as the transaction ID for every command it sends and increment the counter afterward. The counter may roll over to 0x0000 after it reaches 0xFFFF.
- If the accessory receives no response to a command that it has sent, it may send another command with the same payload, but the transaction ID must be different.
- The accessory must ignore any response from the Apple device whose transaction ID does not match that of a previous command it has sent.
- The Apple device will ignore any response from the accessory whose transaction ID does not match that of a previous command it has sent.

Note: The Apple device does not increment transaction ID values during authentication. The Accessory must continue to respond to Apple device commands with the transaction ID included in the Apple device's command.

Enabling and Disabling Transaction ID Support

Accessories must enable their support for transaction IDs according to the following rules:

- Support for transaction IDs must be enabled upon the rising edge of power from the Apple device on pin 13 (Accessory Power) of the 30-pin connector (see *MFi Accessory Hardware Specification*).
- Support for transaction IDs must be enabled before the accessory sends a `StartIDPS` command.

The accessory must continue to enable its transaction ID support for all iAP commands until it encounters one of the following situations:

- Support for transaction IDs must be disabled upon receipt of a General lingo `ACK` command without a transaction ID. Such commands have a payload length value (byte 2) of either 0x04 or 0x08.
- Support for transaction IDs must be disabled upon receipt of a `RequestIdentify` command, but enabled again before the accessory sends a subsequent `StartIDPS` command.
- Support for transaction IDs must be disabled before sending an `IdentifyDeviceLingoes` command.

Note: When responding to an accessory's command containing a transaction ID, an Apple device with older firmware that does not support transaction IDs will send an `ACK` command with a Bad Parameter status (0x04) and no transaction ID.

Apple Device Acknowledgment of Transaction ID Commands

To support commands with transaction IDs, the General lingo `ACK` command has been expanded to five formats, listed in [Table B-1](#) (page 526). The new command details are presented in "[Command 0x02: ACK](#)" (page 83).

Table B-1 Forms of the General lingo `ACK` command

Transaction ID	Command pending	Payload length	Authentication	Format
No	No	0x04	None	Table 2-12 (page 84)
Yes	No	0x06	Version 2.0B	Table 2-13 (page 84)
No	Yes	0x08	None	Table 2-15 (page 86)
Yes	Yes	0x0A	Version 2.0B	Table 2-16 (page 86)
Yes	No	0x0C	Version 2.0B	Table 2-17 (page 87)

Applicability of Transaction IDs

When transaction IDs are enabled, all iAP commands must use them. If the description of the command in this specification does not include transaction ID fields, the developer must add them after the Command ID field in every command packet, and every packet length byte must be increased by 2. There are three exceptions to this requirement; the following commands must omit transaction IDs even though they are enabled for all other commands:

`RequestIdentify` (General lingo command 0x00)
`Identify` (General lingo command 0x01)

IdentifyDeviceLingoes (General lingo command 0x13)

After sending one of these commands the accessory must disable transaction ID support, as described in "Enabling and Disabling Transaction ID Support" (page 525), until it sends a StartIDPS command.

Examples of Transaction IDs

Table B-2 (page 527) through Table B-5 (page 529) illustrate the use of transaction IDs in some typical iAP command flows.

Table B-2 Example of IDPS and authentication

Step	Accessory command	Apple device command	Comment
1	StartIDPS		Transaction ID = 0x0001 (accessory counter starts counting).
2		ACK	Transaction ID = 0x0001; response to Step 1
3	SetFIDTokenValues		Transaction ID = 0x0002.
4		RetFIDTokenValueACKs	Transaction ID = 0x0002; assume some token value was not acknowledged.
5	SetFIDTokenValues		Transaction ID = 0x0003; retry command with a different transaction ID
6		RetFIDTokenValueACKs	Transaction ID = 0x0003; response to Step 5.
7	EndIDPS		Transaction ID = 0x0004.
8		IDPSStatus	Transaction ID = 0x0004; response to Step 7
9		GetDevAuthentication-Info	Transaction ID = 0x0001; first command initiated by the Apple device (Apple device counter starts counting).
10	RetDevAuthentication-Info		Transaction ID = 0x0001; response to Step 9.
11		AckDevAuthentication-Info	Transaction ID = 0x0001; response to Step 10.

Table B-3 Example of entering Remote UI mode

Step	Accessory command	Apple device command	Comment
1	StartIDPS		Transaction ID = 0x0001 (accessory counter starts counting).
2		ACK	Transaction ID = 0x0001; response to Step 1
3	SetFIDTokenValues		Transaction ID = 0x0002.
4		RetFIDTokenValueACKs	Transaction ID = 0x0002; response to Step 3.
5	EndIDPS		Transaction ID = 0x0003.
6		IDPSStatus	Transaction ID = 0x0003; response to Step 5.
Enter background authentication state; see "Authentication" (page 71).			
7	EnterRemoteUIMode		Transaction ID = 0x0004.
8		ACK	Transaction ID = 0x0004; response to Step 7: command pending.
9		ACK	Transaction ID = 0x0004; sent after the Apple device has entered Remote UI mode. Note that the same transaction ID value is used.

Table B-4 Example of getting track artwork data

Step	Accessory command	Apple device command	Comment
1	StartIDPS		Transaction ID = 0x0001 (accessory counter starts counting).
2		ACK	Transaction ID = 0x0001; response to Step 1
3	SetFIDTokenValues		Transaction ID = 0x0002.
4		RetFIDTokenValueACKs	Transaction ID = 0x0002; response to Step 3.
5	EndIDPS		Transaction ID = 0x0003.
6		IDPSStatus	Transaction ID = 0x0003; response to Step 5.
Enter background authentication state; see "Authentication" (page 71).			
7	GetTrackArtworkData		Transaction ID = 0x0004.

Step	Accessory command	Apple device command	Comment
8		RetTrackArtworkData	Transaction ID = 0x0004; Response is divided into multiple packets with the same transaction ID.
9		RetTrackArtworkData	Transaction ID = 0x0004.
10		RetTrackArtworkData	Transaction ID = 0x0004; Last packet returning artwork data.

Table B-5 Example of sending notifications

Step	Accessory command	Apple device command	Comment
1	StartIDPS		Transaction ID = 0x0001 (accessory counter starts counting).
2		ACK	Transaction ID = 0x0001; response to Step 1
3	SetFIDTokenValues		Transaction ID = 0x0002.
4		RetFIDTokenValueACKs	Transaction ID = 0x0002; response to Step 3.
5	EndIDPS		Transaction ID = 0x0003.
6		IDPSStatus	Transaction ID = 0x0003; response to Step 5.
7		iPodNotification	Transaction ID = 0x0001; first command initiated by the Apple device (Apple device counter starts counting).
8		iPodNotification	Transaction ID = 0x0002; second command initiated by the Apple device.
9	RdsReadyNotify		Transaction ID = 0x0004; next accessory transaction ID.
10	RdsReadyNotify		Transaction ID = 0x0005.

Multisection Data Transfers

Three Location lingo commands may need to transfer amounts of data larger than the data receiver's maximum incoming packet size: `RetDevData`, `SetDevData`, and `AsyncDevData`. This appendix specifies how large amounts of data are transferred in sections, using multiple packets of the same command.

Multisection Transfer Process

The packets of commands that can transfer data in multiple sections contain two 16-bit parameters that uniquely identify transferred data sections:

- Parameter `sectCur` is the index number of the data section in that packet. The first packet has index 0x0000.
- Parameter `sectMax` is the index number of the last data section

When a data transfer fits into a single packet, both `sectCur` and `sectMax` must be set to 0x0000. When a command transfers data in multiple sections, the transaction ID of the command that requested the transfer must be used for all response sections and the `sectCur` value of each serial packet must increment by 0x0001. For optimal data throughput, data sections must be the maximum size permitted by the receiver, less any command header overhead. The last data section may be smaller than the maximum size if the total data transfer size is not a multiple of the section size.

An accessory may either send or receive multisection data. When the accessory sends multisection data to an Apple device, using `RetDevData` or `AsyncDevData`, it must wait after each packet for the Apple device to reply with an `iPodACK` command, as detailed in [Multisection iPodACK Command](#) (page 532). When the Apple device sends multisection data to an accessory, using `SetDevData`, the accessory must reply with a `DevACK` command after each packet, as detailed in [Multisection DevACK Command](#) (page 532).

Regardless of whether the accessory is sender or receiver, the acknowledgment command must have the same transaction ID as the data transfer command. When acknowledging any packet before the last one, the acknowledgment must also return the packet's `sectCur` value plus an `ackStatus` of Section Received OK (0x13) if the section was transferred successfully. When the last section has been received successfully, the receiver must send an `ackStatus` of Command OK (0x00), without a `sectCur` value, to indicate that the entire data transfer has finished.

If an error occurs during a multisection transfer, the receiver must send an acknowledgment with a nonzero `ackStatus` other than 0x13. This means the transfer has failed. The sender must stop sending data sections with the same transaction ID. If the receiver does not acknowledge a multisection packet within 400 ms, the transfer has also failed; the sender must send the receiver an acknowledgment with the transfer's transaction ID and a Command Timeout (0x0F) status. When a transfer fails, the sender can try to send the same data again, starting from the beginning, with a different transaction ID.

Multisection iPodACK Command

Direction: Apple device to Accessory

An iPodACK command is sent by the Apple device in response to each multisection data transfer packet sent by RetDevData or AsyncDevData. The Apple device may send either of two forms of the iPodACK packet:

- When any section before the last section has been received successfully, the Apple device sends the packet shown in [Table C-1](#) (page 532). The accessory may send a new packet immediately.
- When the last section has been received successfully, the Apple device sends the packet shown in [Table 3-309](#) (page 402) with `ackStatus = 0x00`, indicating that the multisection data transfer is complete. See [Command 0x80: iPodACK](#) (page 401) for details of this form of the iPodACK command. The Apple device also sends an acknowledgment of this form, with a nonzero `ackStatus`, when an error has occurred in receiving any packet.

Table C-1 Multisection iPodACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x80	Command ID: iPodACK
5	0xNN	transID [bits 15:8]: Transaction ID of command being acknowledged; see "Using Transaction IDs" (page 525)
6	0xNN	transID [bits 7:0]
7	0x13	ackStatus: Section received successfully
8	0xNN	cmdIDorig: Original command ID for which this response is being sent.
9	0xNN	sectCur [Bits 15:8]: Section index for which this response is being sent.
10	0xNN	sectCur [Bits 7:0]
11	0xNN	Checksum

Multisection DevACK Command

Direction: Accessory to Apple device

The accessory must send a DevACK command in response to each multisection data transfer packet sent by the Apple device via SetDevData. The accessory may send either of two forms of the DevACK packet:

- When it has successfully received any section before the last section, the accessory must send the packet shown in [Table C-2](#) (page 533). The Apple device will send a new packet immediately.
- When it has successfully received the last section, the accessory must send the packet shown in [Table 3-286](#) (page 386) with `ackStatus = 0x00`, indicating that the multisection data transfer is complete. See [Command 0x00: DevACK](#) (page 386) for details of this form of the DevACK command. The accessory must also send an acknowledgment of this form, with a nonzero `ackStatus`, whenever it has detected an error in receiving any packet.

Table C-2 Multisection DevACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x00	Command ID: DevACK
5	0xNN	transID [bits 15:8]: Transaction ID of command being acknowledged; see "Using Transaction IDs" (page 525)
6	0xNN	transID [bits 7:0]
7	0x13	ackStatus: Section received successfully
8	0x08	cmdIDOrig: ID of command (SetDevData) for which this response is being sent.
9	0xNN	sectCur [Bits 15:8]: Section index for which this response is being sent.
10	0xNN	sectCur [Bits 7:0]
11	0xNN	Checksum

iTunes Tagging

This appendix explains the iTunes tagging feature and specifies the requirements for broadcast reception accessories to support it.

The iTunes Tagging Experience

To experience the capabilities of the iTunes tagging feature, the user of an Apple device typically goes through the following steps:

1. While listening to broadcast music on an accessory for an Apple device, the user hears a song and decides to tag it for future review or purchase by pressing a special button or other user interface element.
2. The accessory, equipped with a tag button or equivalent software action, stores metadata for the currently playing song. The user tags the song without needing to know anything about it or its originating source. The song information is stored in the accessory until an Apple device is connected to it.
3. When an Apple device is connected to it, the accessory transfers the tagged song information to the Apple device. The Apple device stores the data until the Apple device is synched with iTunes.
4. When the Apple device is connected to the user's computer, iTunes imports the tag data, analyzes it, and presents it to the user in the form of a tagged playlist. iTunes displays the song title, artist, album and other information for each tagged song.
5. Using iTunes, the user may save, review or delete any item in the tagged playlist and may easily purchase any of the listed songs through the iTunes store.

Tagging Feature Components

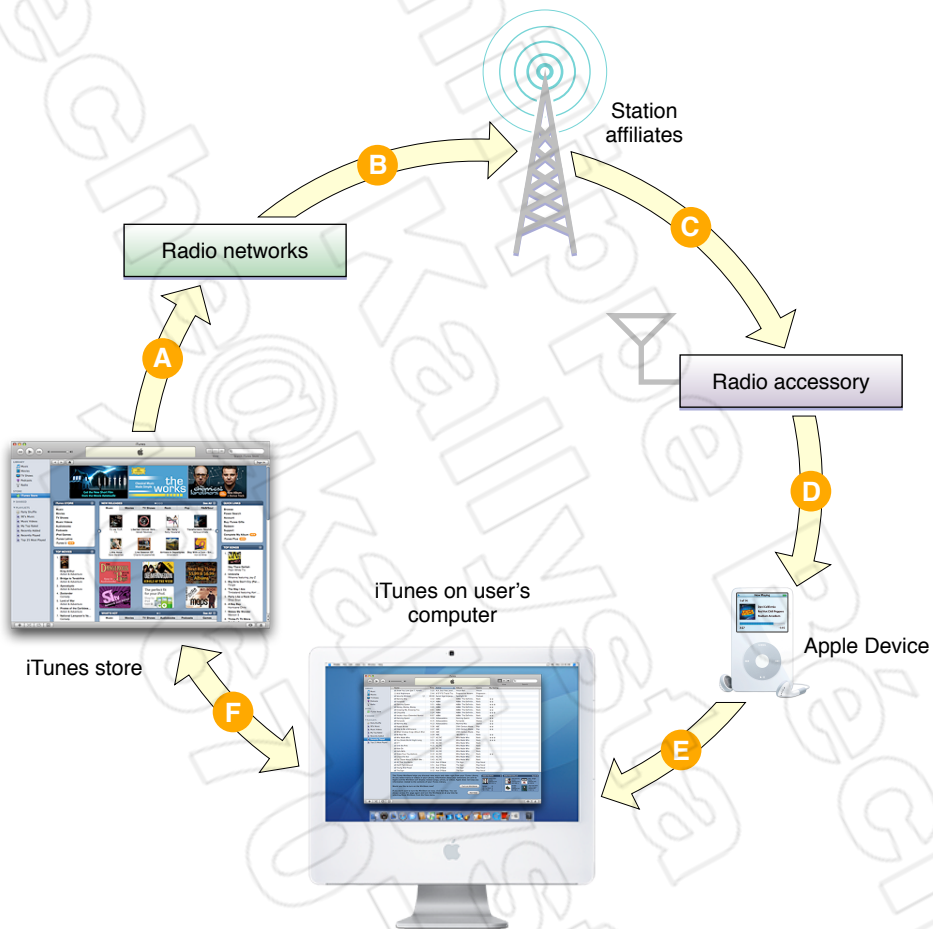
To implement the user experience described in the previous section, the iTunes tagging feature includes the following major components:

- The **iTunes store** creates and publishes unique identifiers for media available in its catalog, and allows users of accessories that support iTunes tagging to purchase tagged songs.
- **Broadcast networks and station affiliates** receive iTunes store data and broadcast it along with program music and other content.
- **Broadcast reception accessories for Apple devices** are designed to enable the tagging feature. These receivers may be portable, used in the home or in a car.
- An **Apple device** is used to transfer tags from broadcast reception accessories to iTunes.

- The **iTunes application** is a desktop media player for managing media content, including newly-discovered music via the iTunes tagging feature.

The data flows among the components of the iTunes tagging feature listed above are diagrammed in [Figure D-1](#) (page 536).

Figure D-1 iTunes tagging feature data flows



Data flow	Description
A	An enterprise partner feed from Apple contains iTunesSongIDs, track names, artists, and album names for the contents of the iTunes store.
B	Network data is sent to the station affiliates, including Apple-supplied data.
C	Affiliates broadcast metadata with each song, containing the song's iTunesSongID, track name, and artist name.
D	The broadcast reception accessory writes XML tag data to a file on an attached Apple device, using the iAP Storage lingo; the Apple device signs the file.

Data flow	Description
E	iTunes verifies and uploads the Apple device's tag files and presents a tagged music playlist to the user.
F	The user clicks a "Buy" button next to the tagged music and downloads the music from the iTunes Store.

Broadcast Reception Accessory Requirements

Broadcast reception accessories that support the iTunes tagging feature described in this specification must perform the following actions:

- Authenticate themselves to the attached Apple device. This requires that the accessory contain an authentication coprocessor supplied by Apple. For technical details, see Apple's *iPod Authentication Coprocessor 2.0B Specification*.
- Conform to the user interface requirements described in "Accessory User Interface" (page 544).
- Provide static storage capacity for at least 50 tags, and provide enough RAM to cache two tags in cases of tag ambiguity (see "Resolving Tag Ambiguity" (page 538)). Accessories should store tags in a data structure and only generate XML when writing files to the Apple device. Tag data may be stored internally in any format convenient to the accessory.
- Generate tag data from the broadcast's metadata. Accessories supporting iTunes Tagging must populate every XML field for which metadata is received.
- Generate files from the tag data and write these files to the attached Apple device. The files must be XML-formatted **plists**, as specified in "Tag Data Writing Process" (page 539). The plist fields in these files are specified in "Data Transfer to the Apple Device" (page 541), and the iAP commands used to write a file to the Apple device are defined in "Lingo 0x0C: Storage Lingo" (page 362).

Capturing and Storing Tag Data

The process of capturing and storing tag data from a broadcast occurs in two steps:

1. When the user presses the tag button while listening to the radio accessory, the accessory stores tag data internally.
2. When the user attaches an Apple device to the radio accessory the stored tag data is written to the Apple device's memory.

Note: Tag data must be written to the Apple device whenever the Apple device is attached, regardless of what mode the accessory may be in. The user must never have to tell the accessory to write tag data nor have to change the accessory's mode to permit tag data to be written.

Resolving Tag Ambiguity

A broadcast tag is deemed ambiguous if the tag button is pressed within 10 seconds before or after a change in the program metadata. In an HD broadcast, a program metadata change is indicated by a Program Service Data (PSD) change; in an FM broadcast, it is indicated by the Event A/B flag. This ambiguity period exists because program metadata changes do not occur exactly on song boundaries, but within 10 seconds before or after that boundary. If the button is pressed less than 10 seconds before the program metadata changes and the new program metadata data is not yet valid, the receiver should generate a nonambiguous tag for the previous data. If a nonambiguous tag is followed by an ambiguous tag of the next song, the result should be two nonambiguous tags. The accessory's UI should not tell the user that a tag is ambiguous.

Note: Tag ambiguity never happens when the user is tagging satellite radio broadcasts.

The accessory must flag ambiguous tags using the `ambiguousTag` plist field. The iTunes application will present a user interface dialog to resolve the ambiguity once the tags have been imported. Accessories should not try to resolve tag ambiguity through their own user interface. Accessories must identify each twin of the ambiguous pair and save this information as a part of the tag data in RAM. Accessories must also identify which twin was active at the time of the tag button press by setting the `ButtonPressed` plist field.

To support tag ambiguity, accessories must store both the current and previous sets of tag data in RAM and update both as program metadata changes occur. A timestamp or timer can be used to mark when either the program metadata changes or the tag button is pressed. The previous and current tags should be stored (when no Apple device is connected) or written to the Apple device with the `ambiguousTag` plist field set to 1 if these two events occur within 10 seconds of each other.

If the tag button is pressed within 10 seconds **before** a program change:

- The accessory timestamps the button press, or starts a 10 second timer when the tag button is pressed, and sets the `ButtonPressed` byte in the current tag data.
- If a program change occurs within the 10 seconds, the tag data in the current slot is moved to the previous slot and the broadcast tag data fills the current slot. The accessory sets the `ambiguousTag` fields for both tags to 1 and writes both tags to the Apple device in a single file.

If the tag button is pressed within 10 seconds **after** a program change:

- The accessory timestamps the program change event, or starts a 10 second timer when the program change occurs, and updates the tag data in the current and previous slots.
- If the tag button is pressed within 10 seconds of the program change, the accessory sets the `ButtonPressed` byte in the current tag data, sets the `ambiguousTag` fields for both tags to 1, and writes both tags to the Apple device in a single file.

Note: Accessories must resolve tag ambiguity (if it exists) before saving tags or writing them to the Apple device. This adds a 10 second delay to the file writing process.

Handling Unknown Metadata Types

Accessories must support features announced after their product releases by supporting the `UnknownData` plist field. This field is used to pass to iTunes unrecognized data types in the broadcast metadata; iTunes parses the data in the `UnknownData` field and handles it appropriately. In HD broadcasts, unknown metadata may occur in the PSD's Unique File Identifier Data (UFID) fields; in FM broadcasts, it may be passed by the iTunes ODA. On encountering unrecognized ID types in a broadcast, iTunes Tagging accessories must encode the entire unknown data field (the complete UFID in the case of HD and the whole iTunes ODA in the case of FM) in `base64` format (64 bytes) and write it to the Apple device as `UnknownData`.

The current HD radio UFID data ID types are listed in [Table D-1](#) (page 539). The current FM tag element types are listed in [Table D-9](#) (page 552).

Table D-1 HD UFID data ID types

ID type	Description
0x3031 or "01"	iTunesSongID
0x3032–0x3033 or "02"–"03"	Reserved
0x3034 or "04"	iTunesAffiliateID
0x3035 or "05"	iTunesStorefrontID
0x3036 or "06"	The contents of the <code>stationURL</code> field is a <code>PodcastFeedURL</code> , not a station URL.

Note: The data in the HD radio UFID Identifier field is encoded in conformance with specification ISO-8859-1 and is represented as strings. Refer to iBiquity Document 2174 for more information about parsing the HD UFID data field.

Tag Data Writing Process

Each tagged track file written to an Apple device consists of an extensible XML dictionary of key-value pairs, formatted as a Mac OS X Core Foundation property list (**plist**). This format is widely used in Mac OS X and can be easily generated and parsed on other platforms. The format is documented at developer.apple.com/documentation/CoreFoundation/Conceptual/CFPropertyLists/index.html. The plist document type definition (DTD) is available at www.apple.com/DTDs/PropertyList-1.0.dtd. Listings of typical of plist files can be found in "Sample Tag Files" (page 562).

Note: Mac OS X plist files are UTF-8 encoded; the PSD data sent by HD radio broadcasters is ISO-8859-1 encoded. The radio accessory must convert the ISO-8859-1 data to UTF-8 before writing the plist file to the Apple device. If the plist file contains ISO-8859-1 characters in the range 0x80 to 0xFF, iTunes cannot parse the file properly and the tag will be lost. The XML markup delimiters `&`, `<`, and `>` must be written as `&`, `<`, and `>`; —otherwise iTunes cannot open the file.

After authenticating itself to the attached Apple device and using `GetiPodOptionsForLingo` to determine that the Apple device supports the iTunes Tagging option, an accessory typically uses the command sequence shown in [Table D-2](#) (page 540) to write a plist file to the Apple device. For details of these commands, see ["Lingo 0x0C: Storage Lingo"](#) (page 362). For a sample sequence of actual commands, see [Table D-6](#) (page 546).

Table D-2 Typical plist writing command sequence

Radio accessory	Attached Apple device
<code>GetiPodCaps</code>	
	<code>RetiPodCaps</code>
<code>GetiPodFreeSpace</code>	
	<code>RetiPodFreeSpace</code>
<code>OpeniPodFeatureFile</code>	
	<code>RetiPodFileHandle</code>
<code>WriteiPodFileData</code> (as many times as needed)	
	<code>iPodACK</code> for each <code>WriteiPodFileData</code> command
<code>CloseiPodFile</code>	
	<code>iPodACK</code> of <code>CloseiPodFileData</code>

The accessory starts by sending a `GetiPodCaps` command to retrieve the Apple device's storage lingo capabilities. The Apple device responds with the `RetiPodCaps` command, which contains the following data:

- `totalSpace`: the amount of free storage on the Apple device.
- `maxFileSize`: the largest possible size of a file on the Apple device.
- `maxWriteSize`: the largest amount of data that can be written to the Apple device in a single `WriteiPodFileData` command.
- `majorVersion` and `minorVersion`: the version number of the Storage lingo protocol (currently 1.1).

Accessories are required to honor the Apple device's `maxWriteSize` limitation; they must never send a `WriteiPodFileData` command with a data payload larger than the Apple device's `maxWriteSize` value.

The accessory should also check the Apple device's free space before creating a file, to ensure that there is enough free space to write the file. A tag takes approximately 1 KB of data space, so the accessory should verify that the Apple device has 1 KB of free space for each tag it intends to write. See ["Note"](#) (page 363). The accessory should also warn the user through its user interface when the Apple device does not have enough free space; see ["Accessory User Interface"](#) (page 544) for details.

To create a file on the Apple device, the accessory sends an `OpeniPodFeatureFile` command. This command returns a file handle that is used to write data and to close the file. The accessory must send a `WriteiPodFileData` command to write data to the open file. The size of the `WriteiPodFileData` payload is determined by the Apple device's `maxWriteSize`; it may take several `WriteiPodFileData` commands to write an entire file. The Apple device responds to each `WriteiPodFileData` command with an `iPodACK` command containing the status of the last write. Accessories must verify that each individual write succeeded before sending the next `WriteiPodFileData` command.

Only one tagging file may be opened at a time. Each time a tagging file is opened, a new file is created in `/iPod_Control/Device/Accessories/Tags/` on the Apple device (this location may change in future releases). You can look at the files in this directory to confirm that the data you wrote using `WriteiPodFileData` was transferred correctly.

The accessory should close the file using the `CloseiPodFile` command as soon as all the plist data has been written to the Apple device. The accessory should verify that the Apple device closed the file properly by checking the status of the `iPodACK` command sent in response to the `CloseiPodFile` command. Accessories should never delete tag data stored locally until the `CloseiPodFile` acknowledgment status has been verified.

Data Transfer to the Apple Device

[Table D-3](#) (page 541) shows all the plist fields an accessory may write to an Apple device. Subsets of these fields for specific broadcasting types are listed in ["FM Radio Tagging"](#) (page 550), ["HD Radio Tagging"](#) (page 560), ["Satellite Radio Tagging"](#) (page 561), and ["Other Broadcast Tagging"](#) (page 562).

The plist writing process must follow these rules:

- All fields of integer type must be expressed in decimal numbers. Hexadecimal and other non-decimal numbers may not be used.
- Strings in the `Name`, `Artist`, `Album`, `Genre`, and `StationURL` fields must not be truncated; if 64 bytes are received, all 64 must be written. The `StationURL` field may contain 128 bytes.

Table D-3 Plist fields written to the Apple device

Name	Type	Description	Source
<code>MajorVersion</code>	integer	The major revision level of the file format. This number increments by 1 if there are changes or additions that break compatibility with all prior versions. The current major revision level is 1.	Accessory

Name	Type	Description	Source
MinorVersion	integer	The minor revision level of the file format. This number increments by 1 if there are changes or additions that preserve compatibility with prior versions that have the same major revision level. The current minor revision level is 1.	Accessory
ManufacturerID	positive integer (decimal format)	A product-specific ID number (see note, below). This number is different from the number assigned to Nike + iPod Cardio accessories, described in Table E-5 (page 577).	Accessory
ManufacturerName	string	The manufacturer-assigned user-visible name for the manufacturer.	Accessory
DeviceName	string	The manufacturer-assigned user-visible name for the accessory.	Accessory
MarkedTracks	array	An array of one or more dictionaries, specifying the data for each tagged track.	Accessory
AmbiguousTag	integer	1 to mark an ambiguous track; 0 otherwise.	Accessory
ButtonPressed	integer	1 to designate which twin in an ambiguous pair was active when the tag button was pressed; 0 otherwise.	Accessory
Name	string	The name of the track.	Broadcast Metadata
Artist	string	The name of the artist.	Broadcast Metadata
Album	string	The name of the album.	Broadcast Metadata
Genre	string	The genre of the track.	Broadcast Metadata
iTunesSongID	integer	The iTunes song identifier.	Broadcast Metadata
iTunesStorefrontID	integer	The iTunes storefront identifier.	Broadcast Metadata
StationFrequency	string	The station's frequency, expressed only by digits and an optional decimal point.	Accessory
StationCallLetters	string	The station identifier.	Broadcast Metadata

Name	Type	Description	Source
TimeStamp	date	The date and time at which the tagged track was broadcast, in ASCII timestamp format (e.g., 2008-04-16T00:35:42Z). The time zone must be UTC; iTunes converts it into local time.	Broadcast Metadata
PodcastFeedURL	integer	A value of 1 if the <code>StationURL</code> points to a podcast, 0 otherwise.	Broadcast Metadata
StationURL	string	The URL of the station, or of a podcast if <code>PodcastFeedURL</code> is 1.	Broadcast Metadata
ProgramNumber	integer	The station's multicast channel in the range 1 to 8, where 1 indicates the main program.	Broadcast Metadata
iTunesAffiliateID	string	The iTunes affiliate ID, as broadcast (not applicable to FM broadcasts—leave blank).	Broadcast Metadata
iTunesStationID	string	The iTunes station ID, as broadcast.	RDS only
UnknownData	data	A field used to pass unknown UFID data to iTunes, defined only for HD and FM broadcasts. See "Handling Unknown Metadata Types" (page 539).	Broadcast Metadata

Note: The `ManufacturerID` number is part of a unique product ID assigned by Apple. The first part of the Apple-assigned ID identifies the licensee's account and the second part is the specific product number to be entered in this field. For example, if Apple assigns a product ID of 678999-12345, the `ManufacturerID` field must be set to 12345. To obtain a product ID number, the licensee must submit a product plan to Apple's MFi portal. Apple will respond by assigning a unique number to every separate product (i.e., every SKU).

The radio accessory normally writes a tag to the Apple device each time the user presses the tag button. However, the tag should not be written if tag ambiguity exists and has not been resolved (see ["Resolving Tag Ambiguity"](#) (page 538)). The opening, writing, and closing of the file should happen in one continuous action to insure that no tag information is lost. Accessories should not keep files open any longer than absolutely necessary. Do not append tags to open files; write a new file to the Apple device each time the user presses the tag button. The accessory design must allow tagging while data is being written to the Apple device.

The accessory must create a file for each tag saved when the Apple device is connected and the tag is not ambiguous. The accessory must write ambiguous tags to the same file when the Apple device is connected and the tags are ambiguous. The accessory must write all tags stored locally to one file the next time an Apple device is connected, including all ambiguous tags.

Note: The fields `MajorVersion`, `MinorVersion`, `ManufacturerID`, `ManufacturerName`, `DeviceName`, `MarkedTracks`, `Name`, and `Artist` are required to store a tag file. The `iTunesSongID` field is desirable but not required.

An accessory must never write a tag more than once. If the user presses the tag button multiple times during the same song, the accessory must write the tag to the Apple device after the first button press and then filter out redundant tags by setting a flag to indicate that the broadcast tag data has been written to the Apple device.

The accessory must store tags internally when no Apple device is connected. If the accessory runs out of storage space, it must prompt the user to connect an Apple device. When tags are stored locally, the accessory must write all tag data to a single file the next time an Apple device is docked. The `MajorVersion`, `MinorVersion`, `ManufacturerName`, `ManufacturerID`, and `MarkedTracks` fields need to be written only once per file when writing multiple tags to a single file. The rest of the tag data is written in a tag array with each tag designated by the `<dict>` key. "Sample Tag Files" (page 562) shows an example of multiple tags in a single file.

Before it deletes any tag stored locally, the accessory must verify that the Apple device has sent an `iPodACK` command in response to each `WriteiPodFileData` command and the `CloseiPodFile` command. The accessory must notify the user that the tags were successfully written to the Apple device.

Accessory User Interface

Table D-4 (page 544) lists the user interface elements that must be used for the tagging feature in a radio accessory. The table shows three sections: Required UI, Optional UI, and UI not allowed. Table D-5 (page 545) lists user interface messages that may appear on the accessory's display.

Table D-4 Tagging feature user interface implementation

Action		Implementation	
		Speaker	Head unit
Required UI	Indicate tag available	Button (LED) illuminates	Button or logo/type appears
		Logo/type appears (LCD)	Button color or logo/type changes
	Indicate tag captured	Button LED blinks	Button blinks
		Button LED changes color	Button changes color
		Logo/type blinks (LCD)	Graphic
		Message on LCD	Message on screen
		Audible feedback	Audible feedback
	Indicate accessory memory full	Message (LCD)	Message
		Audible feedback	Audible feedback

Action		Implementation	
		Speaker	Head unit
			Graphic
	Indicate Apple device memory full	Message (LCD)	Message
		Audible feedback	Audible feedback
Optional UI	Indicate write to Apple device	Button LED blinks twice	Button blinks twice
		Message on LCD	Message on screen
		Audible feedback	Audible feedback
	Indicate tag not captured	LED color changes	Button color changes
		Message on LCD	Message on screen
		Audible feedback	Audible feedback
	Tags remaining	Message on LCD	Message on screen
UI not allowed	Resolving tag ambiguity		
	Choosing whether or not to write tags to the Apple device		

Table D-5 Tagging feature UI text messages

Condition	Message
Capturing tags	"Tag available"
	"Tag stored"
	"Tag stored. XX remaining."
Accessory memory full	"Memory full. Connect iPod."
	"Connect iPod to transfer tags."
Apple device memory full	"iPod full. Tags cannot be stored."
Write to Apple device	"Tags transferred to iPod."
	"Tags saved to iPod. XX remaining."

Note: These message texts are preliminary and are provided for guidance only. The specific texts to be displayed are expected to change as the tagging feature user interface is further defined.

The accessory should not have a UI function that deletes individual tags; this is done by iTunes. A function to delete all tags is acceptable as a way to restore the accessory to its factory settings.

Tag Button Text

The button on the radio accessory used to tag a song should bear the word “Tag.” The word “Tag” should be presented in a manner consistent with the text on the accessory’s other buttons (the same font, weight, capitalization, color, illumination, and so on).

Sample Command Sequence

Table D-6 (page 546) shows a sample sequence of commands that implements radio tagging in an accessory.

Table D-6 Radio tagging command sequence

Step	Accessory command	Apple device command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> ■ If the accessory receives no ACK command within 500 ms, it may retry Step 1 at 500 ms intervals as long as Apple Device Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them. ■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the Apple device does not support IDPS. Within 800 ms the accessory must send an IdentifyDeviceLingoes command with all fields set to 0xFF. See "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). A sample command sequence is listed in Table F-46 (page 633). ■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1. 			
3	GetiPodOptions-ForLingo		getting options for General Lingo
4		RetiPodOptions-ForLingo	returning options of 000000000003F3FF (Line out usage Video output NTSC video signal format PAL video signal format Composite video out connection S-Video video out connection Component video out connection Closed captioning (video) Video aspect ratio 4:3 (fullscreen) Video aspect ratio 6:9 (widescreen) reserved app communication capable iPod notifications) for General Lingo

Step	Accessory command	Apple device command	Comment
5	GetiPodOptions-ForLingo		getting options for Storage Lingo
6		RetiPodOptions-ForLingo	returning options of 0000000000000003 (iTunes tagging Nike + iPod cardio equipment) for Storage Lingo
7	SetFIDTokenValues		setting 8 FID tokens ; IdentifyToken = (lingoes: 0/12 options 0x00000002 device ID 0x00000200); AccCapsToken = 0x00000000000000805; AccInfoToken = Acc name (Radio); AccInfoToken = Acc FW version (v1.0.1); AccInfoToken = Acc HW version (v1.0.0); AccInfoToken = Acc manufacturer (Radio Manufacturer); AccInfoToken = Acc model number (M78901LL/Z); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'selected')
8		RetFIDTokenValueACKs	8 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; iPodPreferenceToken = (line out usage) accepted
9	EndIDPS		status 'finished with IDPS; proceed to authentication'
10		IDPSStatus	status 'ready for auth'
11		GetDevAuthentication-Info	no params
12	RetDevAuthentication-Info		returning auth protocol v2.0; current section index: 0; maximum section index: 1; cert data ...
13		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
14	RetDevAuthentication-Info		returning auth protocol v2.0; current section index: 1; maximum section index: 1; cert data ...

Step	Accessory command	Apple device command	Comment
15		AckDevAuthentication-Info	acknowledging 'auth info supported'
16		GetDevAuthentication-Signature	offering challenge ...
17	RetDevAuthentication-Signature		returning signature ...
18		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
19	GetiPodCaps		no params
20		RetiPodCaps	returning 'total space 2147483648; max file size 1073741824; max write size 500; v1.02'
21	GetiPodFreeSpace		no params
22		RetiPodFreeSpace	returning 130023424 bytes
23	OpeniPodFeatureFile		opening feature type 'Radio Tagging'
24		RetiPodFileHandle	returning file handle 0
25	WriteiPodFileData		writing 184 bytes at offset 0 and handle 0: <?xml version=""1.0" encoding=""UTF-8""?> <!DOCTYPE plist PUBLIC ""-//Apple Inc//DTD PLIST 1.0//EN"" "http://www.apple.com/DTDs/PropertyList-1.0.dtd""> <plist version=""1.0""> <dict>
26		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
27	WriteiPodFileData		writing 290 bytes at offset 185 and handle 0: <key>DeviceName</key> <string>IES2</string> <key>MajorVersion</key> <integer>1</integer> <key>MinorVersion</key> <integer>0</integer> <key>ManufacturerID</key> <integer>17</integer> <key>ManufacturerName</key> <string>Polk Audio</string> <key>MarkedTracks</key> <array> <dict>

Step	Accessory command	Apple device command	Comment
28		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
29	WriteiPodFileData		writing 472 bytes at offset 476 and handle 0: <key>Name</key> <string>The Rising</string> <key>Artist</key> <string>Bruce Springsteen</string> <key>Album</key> <string>The Rising</string> <key>StationURL</key> <string>http://www.hdradio.com</string> <key>iTunesSongID</key> <integer>192903160</integer> <key>StationCallLetters</key> <string>IHDR</string> <key>StationFrequency</key> <string>88.1 FM</string> <key>StreamID</key> <string>2</string> <key>ProgramType</key> <string>Classic Rock</string>
30		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
31	WriteiPodFileData		writing 210 bytes at offset 949 and handle 0: <key>TimeStamp</key> <date>2009-06-01T09:45:57Z</date> <key>iTunesStorefrontID</key> <integer>143441</integer> <key>iTunesPlaylistID</key> <integer>192901525</integer> </dict> </array> </dict> </plist>
32		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
33	CloseiPodFile		closing file with handle 0
34		iPodAck	acknowledging 'Success' to command 'Storage Lingo::CloseiPodFile' with file handle 0

FM Radio Tagging

FM radio uses the RDS/RBDS subcarrier system to broadcast data, in addition to its audio content. For information about RBDS, see the *United States RBDS Standard, NRSC-4-A*, available at www.nrsstandards.org. FM tagging data is broadcast inside two ODAs (Open Data Applications): a RadioText Plus (RT+) ODA and a private iTunes Tagging ODA.

The FM radio accessory must capture the data streams from both these ODAs and write their information to plist files, as described in "Capturing and Storing Tag Data" (page 537). During this process, the accessory must populate every plist field listed in Table D-7 (page 550) for which it receives metadata from the radio broadcast. For an FM tag to be usable, its plist entry must provide valid data for at least the `Title` and `Artist` fields.

Table D-7 Required plist fields for FM

Name	Type	Description	FM Source
Name	string	The name of the track.	RT+ type: ITEM.TITLE
Artist	string	The name of the artist.	RT+ type: ITEM.ARTIST
Album	string	The name of the album.	RT+ type: ITEM.ALBUM
Genre	string	The genre classification of the track.	RT+ type: ITEM.GENRE
iTunesSongID	integer	The iTunes song identifier.	iTunes ODA: iTunesSongID
iTunesStorefrontID	integer	The iTunes storefront identifier.	iTunes ODA: iTunesStorefront
StationFrequency	string	The station's frequency, expressed only by digits and an optional decimal point.	Accessory
StationCallLetters	string	Station call letters can be extracted from the RDS PI field or through the RT+ STATIONNAME.SHORT and STATIONNAME.LONG classes.	RT+ type: STATIONNAME.SHORT; STATIONNAME.LONG
TimeStamp	date	The date and time at which the tagged track was broadcast, in ASCII timestamp format (e.g., 2008-04-16T00:35:42Z). The time zone must be UTC; iTunes converts it into local time.	RDS: Group 4A
PodcastFeedURL	integer	Indication that the broadcast has an associated podcast. If RT+ type INFO.URL is filled, the broadcast should be marked as having an associated podcast. Set to 1 if there is a Podcast URL stored in the StationURL field.	N/A
StationURL	string	The URL of the broadcast station as taken from the RT+ type PROGRAMME.HOMEPAGE, or of a podcast if PodcastFeedURL is 1 as taken from the RT+ type INFO.URL.	RT+ type: PROGRAMME.HOMEPAGE; INFO.URL

Name	Type	Description	FM Source
ProgramNumber	integer	The station's broadcast subchannel.	RT+ type: P PROGRAMME.SUBCHANNEL
iTunesStationID	string	The iTunes station ID, as broadcast. The lower byte of the PI code concatenated with the Region ID code.	RDS: PI code and Region ID code
UnknownData	data	A field used to pass unknown iTunes Tagging ODA data types to iTunes. See " Handling Unknown Metadata Types " (page 539).	iTunes ODA

The RT+ ODA

RadioText Plus (RT+) is a technology that tags RDS RadioText (RT) messages so specific content can be retrieved from them. RT+ tags are transmitted in RDS broadcasts as an ODA with an Application ID (AID) of 0x4BD7. You can obtain the RT+ specification from the RDS Forum at www.rds.org.uk/.

Each RT+ tag contains three elements:

- The RT content type
- The position inside the RT message of the first character of content to be retrieved
- The length of the content to be retrieved.

The RT+ content types used for iTunes FM radio tagging are listed in [Table D-8](#) (page 551). The corresponding plist fields sent to the Apple device are described in [Table D-3](#) (page 541).

Table D-8 RT+ content types

RT+ type	iTunes plist field
ITEM.TITLE	Name
ITEM.ARTIST	Artist
ITEM.ALBUM	Album
ITEM.GENRE	Genre
STATIONNAME.SHORT, STATIONNAME.LONG	StationCallLetters
INFO.URL	PodcastFeedURL
PROGRAMME.HOMEPAGE	StationURL
PROGRAMME.SUBCHANNEL	ProgramNumber

The iTunes Tagging ODA

The iTunes ODA used for FM broadcasts encodes tag information as **elements**. Currently 6 element types are defined and another 10 are reserved for future use, as shown in [Table D-9](#) (page 552).

Table D-9 FM tag element types

ID type	Description
0x0	Event Control
0x1	iTunes Song ID
0x2	iTunes Artist ID
0x3	Reserved
0x4	Industry ID
0x5	iTunes Storefront
0x6	Extended ID
0x7-0xF	Reserved

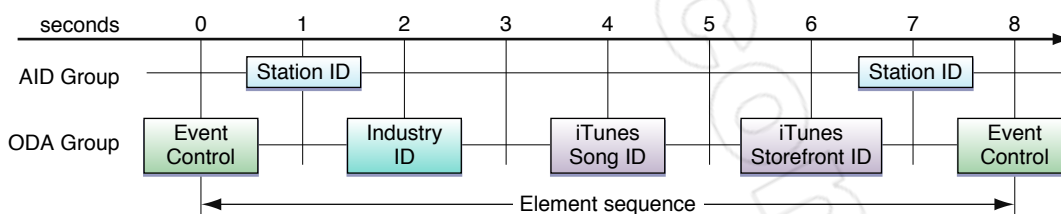
All currently defined FM tag elements carry 32 bits of data and are transmitted using a single ODA application group. If necessary, future versions of the iTunes FM tagging protocol may accommodate larger data types by chaining several ODA groups together.

FM tag elements are normally encrypted before transmission. Because the transmission must be decrypted by the receiver, it is keyed to a composite of values contained in the RDS data. This process is described in ["FM Tag Decryption"](#) (page 555).

A complete iTunes tagging transmission normally consists of several element types in an **element sequence**. The elements in the sequence are transmitted at a relatively low rate (about one element every 2 seconds), so the RDS bandwidth consumed by iTunes tagging is relatively small. When a complete sequence of elements has been transmitted the process repeats itself. [Figure D-2](#) (page 552) shows the transmission timeline of a typical element sequence. In this example, the sequence consists of 4 element types and a complete transmission of the sequence occurs every 8 seconds.

Note: To assure that tag data is always available, the maximum recommended time between transmissions is 10 seconds.

Figure D-2 iTunes tagging element sequence



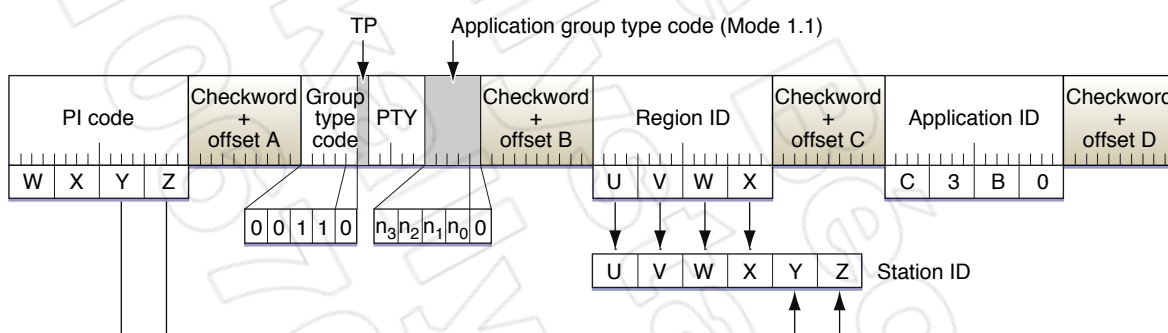
Use of the RDS Group Type 3A

Because the RT+ and FM iTunes tagging data are contained in ODAs, they use the RDS group type 3A, also called the AID (Application ID) Group type. This RDS group type includes 16 message bits for use by each ODA (block C) and indicates the application group type (if any) used for transmission of additional ODA data. Tag data is transmitted in mode 1.1, meaning that the broadcaster allocates an available type A group for transmission of iTunes tag element data. RDS 3A group messages with Application ID 0xC3B0 indicate which RDS group type will be used to carry the iTunes tagging information. The accessory must parse group type 3A messages and save the application group type code (n3-n0), because this indicates the ODA group type on which tag data will be broadcast.

Note: The accessory must parse and save the application group type code for each iTunes-specific type 3A message, because the type code for subsequent iTunes data may change during the broadcast.

The recommended transmission rate for the iTunes tag 3A group is once every 7 seconds, with a maximum repeat delay of once every 10 seconds. Upon tuning to a new frequency, a tagging-enabled FM receiver waits for the reception of a 3A group containing the assigned Application ID number; this indicates that the broadcast includes iTunes tag data. The RDS 3A group structure for iTunes tagging is shown in [Figure D-3](#) (page 553).

Figure D-3 RDS 3A group for iTunes tagging



The Region ID code shown in [Figure D-3](#) (page 553) combines the extended country code (see Annex N of the RBDS specification) and the upper byte of the PI (Program Identification) code: the upper byte "UV" of the Region ID carries the extended country code and the lower byte "WX" mirrors the upper byte of the PI code. The receiver must combine the 16-bit Region ID with the lower byte of the PI code to form a 24-bit Station ID, a guaranteed unique identifier of a particular radio station. The 24-bit Station ID has the byte order UVWXYZ, as shown in the diagram.

Note: It is possible that the upper byte of the PI code may not exactly mirror the lower byte of the Region ID. For this reason, a receiver must construct the Station ID exactly as specified above. Always use the bytes transmitted in Region ID; ignore the upper byte of the PI code.

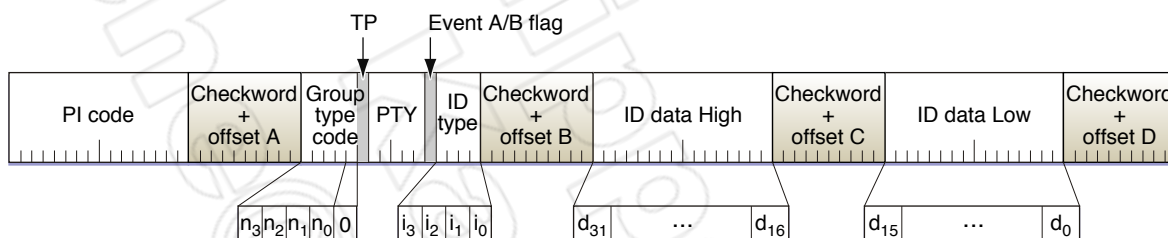
The Station ID is intended for use in affiliate programs. Because affiliate programs normally involve an online merchant, that merchant or an appropriate third party maintains a table translating the Station ID into a more traditional Affiliate ID. For example, a range of Station ID codes may map to a single group identifier for the purposes of an affiliate program.

Tagging Application Group

The iTunes tag element data is transmitted using an RDS type A application group (also called an ODA Group), as shown in [Figure D-4](#) (page 554). The Group Type Code (n3-n0) of this message should correspond to that broadcast as the application group type code, which was part of the Group 3A message with AID 0xC3B0.

[Figure D-4](#) (page 554) depicts the data before encryption at the transmitter and following successful decryption at the receiver. The encryption process scrambles the ID Type (bits i3 to i0) as well as the ID data (bits d31 to d16 and d15 to d0). The remaining data (PI code, group type code, TP flag, PTY and Event A/B flag) are always transmitted in the clear.

Figure D-4 RDS application group for iTunes tagging



The ID Type indicates the specific type of element data carried by an instance of the ODA group. There are 16 possible element types of which 6 are currently defined (see [Table D-9](#) (page 552)). The unassigned elements are available for future use. Types 00000 and 1111 are reserved as special-purpose control elements; all other types are available for content identification codes. Data received with an unassigned element type must be written to an `UnknownData` field in the plist sent to the Apple device; see ["Handling Unknown Metadata Types"](#) (page 539).

The ID data field carries the actual data associated with an element. For all currently-defined numeric identifiers (iTunes Song, iTunes Artist, iTunes Storefront, Extended ID and Industry ID) these bits are interpreted as 32-bit unsigned integer values. The Event Control element uses an alternate definition of the data bits, as detailed in ["FM Tagging Event Control"](#) (page 555). The Event A/B flag is a toggle bit that changes state each time the broadcast content changes. This tells the receiver that it should clear any temporary buffers associated with element data in preparation for reception of a new set of data. In a typical music broadcast, the Event A/B flag toggles at the transition from one song to the next, at the transition from music to non-music content (advertisement break) and at the transition from non-music content back to music. Even if there are no elements associated with a particular piece of content, the A/B flag still toggles at the start of that content to mark the end of previously identified content. In such a situation, an element sequence consisting only of the Event Control element is transmitted as a carrier for the A/B flag.

Normal Mode and Test Mode

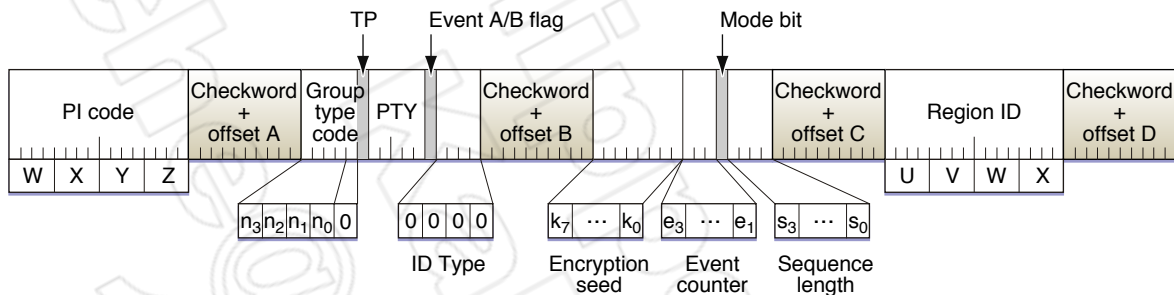
Two modes of operation are defined for the RDS transmission of iTunes tag data: **normal mode** and **test mode**. The Region ID section of Group Type Code 3A messages with AID 0xC3B0 indicate what mode is being used for broadcast. In normal mode the message bits of the AID group carry a 16-bit Region ID and encryption/decryption is enabled. In test mode the message bits are set to zero (0x0000) and encryption/decryption is disabled. Tagging-enabled FM receivers must support both operating modes. In test mode, the Station ID is constructed using Region ID and the PI code lower byte as carried by the Event Control element (See ["FM Tagging Event Control"](#) (page 555)). In test mode it is not possible to construct a Station ID from the 3A group because the Region ID is set to zero.

FM Tagging Event Control

Event Control is a special-purpose element used to indicate a change in the content being broadcast as well as to indicate the overall status of the iTunes tag transmission.

The **event counter** is a 4-bit count that is incremented each time the on-air content changes. After the count reaches 15 it wraps back to 0. The 3 most significant bits of the count (bits e3-e1) are carried in the Event Control element, shown in [Figure D-5](#) (page 555). The least significant bit of the counter (bit e0) is carried in every element type and is called the Event A/B Flag.

Figure D-5 Event Control element



When set to 1, the mode bit shown in [Figure D-5](#) (page 555) indicates that the event currently being broadcast has been successfully mapped to at least one type of content identifier and that identifiers are being transmitted as part of the element sequence. If no mapping exists for the current event, the mode bit is set to 0. The Sequence length bits (bits s3-s0) indicate the number of element types included in the element sequence for the current event. A value of 0000 indicates that the sequence consists only of the Event Control element. The decryption seed (bits k7-k0) is a one-byte value used to seed the decryption system discussed in ["FM Tag Decryption"](#) (page 555). The 16-bit Region ID value transmitted in block C of the AID group is also transmitted as block D of the Event Control element.

Note: The decryption seed must be read from every Event Control message because it may change during the broadcast.

In normal mode, this retransmission lets a receiver quickly verify that Region ID has been received error-free and that element data is being decoded correctly. Retransmission is particularly valuable because the Region ID (as a component of the Station ID) is used as part of the decryption process. In test mode, the Region ID is transmitted only as part of the Event Control element. Retransmission is of less value in test mode because encryption/decryption is disabled.

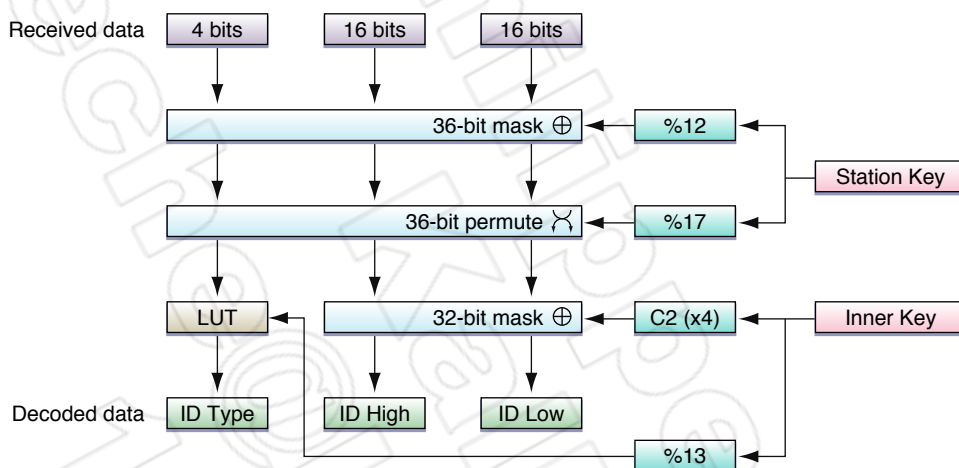
FM Tag Decryption

During the transmission of FM tag elements, 36 bits of raw data are passed into an encryption process, including the 4-bit element type ID and 32 bits of element data (ID High and ID Low). The encryption algorithm is keyed by two values generated from RDS data: the Station Key and the Inner Key. The Station Key is the algebraic sum of all three bytes of the Station ID: $\text{StationKey} = YZ + UV + WX$. The Inner Key is the linear combination (XOR) of three byte values: the 8 least significant bits of the Station Key, a byte formed from

Sequence Length (upper nibble) and Event Counter (lower nibble), and the one-byte Encryption Seed carried by the Event Control element: $\text{InnerKey} = (\text{StationKey} \& 0xFF) \wedge ((\text{SequenceLength} \ll 4) \mid \text{EventCounter}) \wedge \text{EncryptionSeed}$.

The decryption process for FM iTunes tagging elements is diagrammed in [Figure D-6](#) (page 556) and is described below.

Figure D-6 Element decryption process



To decode the received data, it is first necessary to generate the station key, which is the algebraic sum of all three bytes of the Station ID. ($\text{StationKey} = YZ + UV + WX$). The three input bytes and the result are treated as unsigned integers. The Station ID bytes can be recovered from the AID group.

Upon reception of an ODA group, the 36 bits of coded tag data are passed into the decryption process. The first stage of decoding applies 1 of 12 predefined outer masks to the full 36 bits of received data using a linear mixing (XOR) operation. The 12 predefined masks are listed in [Table D-10](#) (page 556). The mask used is selected as the modulus (integer remainder) of the Station Key divided by 12 ($\text{MaskRow} = \text{StationKey} \% 12$).

Table D-10 Outer mask lookup table

Row	Bits 35-32	Bits 31-16	Bits 15-0
0	D	D5D0	E3E1
1	2	6073	04B4
2	C	5C59	80D5
3	E	9C6E	78D0
4	3	9E48	2754
5	4	062E	9DB7
6	B	2924	4436
7	6	F308	3628

Row	Bits 35-32	Bits 31-16	Bits 15-0
8	9	9DE8	27A4
9	5	1C71	CEA0
10	8	B5C7	C134
11	7	679C	30BD

The second stage of decoding applies 1 of 17 predefined permute operations to the full 36 bits of received data. The permutation operation reorders the data at the bit level. The 17 permutation operations are shown in [Table D-11](#) (page 557) and [Table D-12](#) (page 558). The operation used is selected as the modulus of the station key divided by 17 ($\text{PermuteRow} = \text{StationKey} \% 17$).

Table D-11 Permutation lookup table, upper bits

Bits	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18
0	7	11	4	35	3	5	26	2	21	32	6	20	17	0	14	27	9	15
1	13	29	28	12	19	33	32	15	24	27	2	9	4	34	17	8	16	22
2	25	9	24	13	18	11	5	32	22	17	35	16	2	29	6	15	1	19
3	16	20	2	0	28	14	6	25	12	8	7	1	32	24	4	19	33	9
4	10	8	20	7	16	28	33	21	35	1	15	12	25	17	11	22	18	23
5	15	17	16	19	7	3	30	8	28	12	29	34	11	18	25	13	2	31
6	21	31	11	6	34	13	0	33	17	9	26	10	14	7	2	18	22	20
7	4	22	9	15	14	12	1	6	7	21	33	19	35	11	16	29	10	5
8	0	4	35	18	6	7	24	23	11	34	32	3	1	31	22	16	8	30
9	18	2	15	20	5	22	3	19	4	25	31	26	0	28	13	7	29	1
10	3	0	5	9	17	1	8	18	13	31	4	15	6	12	20	14	23	25
11	9	6	1	33	30	32	21	29	19	13	0	22	10	5	7	2	14	12
12	28	24	26	34	33	18	14	5	25	15	10	0	9	32	19	31	17	13
13	31	5	13	14	0	35	17	12	30	29	22	18	24	25	1	21	4	26
14	22	32	23	24	15	16	35	11	6	33	18	14	28	21	29	0	27	34
15	2	30	0	4	13	15	18	9	8	20	28	32	21	10	31	6	11	35
16	19	33	10	2	27	17	22	7	16	3	21	5	30	20	24	35	32	14

Table D-12 Permutation lookup table, lower bits

Bits	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	8	23	29	30	22	19	16	25	24	10	1	33	12	28	34	18	13	31
1	18	10	31	11	26	3	35	6	25	23	5	0	1	30	7	20	14	21
2	10	8	30	27	23	7	31	12	33	3	0	34	20	21	26	14	28	4
3	5	3	34	31	35	30	21	26	22	15	13	29	18	10	11	23	17	27
4	13	14	2	24	30	29	34	19	26	31	32	4	3	27	9	0	5	6
5	20	6	24	26	10	1	0	23	27	21	4	22	14	5	35	33	32	9
6	1	24	28	4	32	5	15	8	23	29	12	25	30	3	27	19	16	35
7	32	27	20	3	34	31	18	13	30	0	23	28	24	2	25	17	26	8
8	21	2	26	29	12	9	20	17	13	33	25	5	10	14	15	28	27	19
9	27	11	16	8	6	17	24	32	35	9	34	30	21	23	14	12	10	33
10	33	19	27	7	2	21	30	28	34	24	10	16	26	22	29	35	11	32
11	34	31	11	23	20	25	8	27	3	4	15	17	28	26	18	16	35	24
12	35	4	21	6	7	23	29	20	8	30	2	3	27	1	16	11	22	12
13	2	33	7	19	16	20	28	3	32	6	8	27	23	34	10	15	9	11
14	3	26	19	1	31	2	4	5	20	17	30	13	9	12	8	10	7	25
15	26	17	5	12	29	16	27	24	19	34	22	23	25	33	1	7	3	14
16	23	1	8	25	4	26	13	15	31	18	29	12	6	11	0	9	34	28

Note: At this stage the outer layer of the encryption system has been reversed and it is now possible to read data from the control elements (upper 4 bits set to 0000 or 1111). The reception and parsing of an event control element (type 0000) is required before the remaining ID oriented elements can be decoded.

The third stage decodes the ID type by passing the upper 4 data bits through a predefined lookup table. It then decodes the ID high and ID low bits by applying an algorithmically-generated inner mask to the lower 32 data bits. The ID lookup is shown in [Table D-13](#) (page 558). The lookup used is selected as the modulus of the inner key divided by 13 ($IDlookupRow = InnerKey \% 13$). The upper 4 data bits select the table column and the decoded ID type is the cell value at the selected row and column.

Table D-13 Element type lookup table

ID	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	6	7	2	9	8	11	4	3	10	13	14	1	12	5	15

ID	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	3	10	4	5	6	7	11	2	12	9	1	13	14	8	15
2	0	9	3	11	6	7	4	1	14	13	12	5	8	2	10	15
3	0	14	1	9	13	4	12	6	10	11	3	8	2	5	7	15
4	0	7	14	8	3	10	2	12	13	1	5	4	11	9	6	15
5	0	13	11	12	1	14	5	8	4	3	6	10	9	7	2	15
6	0	11	4	13	14	2	9	3	7	8	1	6	5	10	12	15
7	0	2	9	7	10	12	14	13	5	4	8	3	6	1	11	15
8	0	8	13	6	2	9	10	5	11	14	4	12	7	3	1	15
9	0	4	12	1	11	13	8	10	9	5	2	7	14	6	3	15
10	0	5	8	14	7	1	3	9	12	6	11	2	10	4	13	15
11	0	10	6	5	12	11	13	2	1	7	14	9	3	8	4	15
12	0	12	5	10	8	3	1	14	6	2	7	13	4	11	9	15

Note: At this point the inner layer of the encryption process for the element type has been reversed and all element types are fully decoded. Decryption of the data bits is accomplished by passing them through the polynomial shown in [Listing D-1](#) (page 559).

To decode ID-oriented elements, it is necessary to generate the inner key. The inner key is a linear mixing (XOR) of three byte values: the 8 least significant bits of the station key, a byte formed from the sequence lLength (upper nibble) and event counter (lower nibble) and the encryption seed value, all carried by the event control element ($\text{InnerKey} = (\text{StationKey} \& 0xFF) \wedge ((\text{SequenceLength} \ll 4) \mid \text{EventCounter}) \wedge \text{EncryptionSeed}$).

The inner mask is a pseudo-random bit pattern output from a linear feedback shift register seeded by the Inner Key. The Baicheva C2 algorithm is used as the generator polynomial. Four calls are made to the random number generator, each of which produce 8 bits of data. The seed value of the first call is the inner key. The result of the first call becomes the upper 8 bits of the inner mask. The result of the first call also becomes the seed value of the second call, and so on. The result of the fourth call becomes the lower 8 bits of the inner mask.

A code listing for the pseudo-random number generator is shown in [Listing D-1](#) (page 559). To build the 32-bit inner mask, four calls are made to the `IDI_C2` function. The initial call passes in the seed value as the parameter `M`. Passing the output of a call (the return value) in as the seed value of a subsequent call produces the required pseudo-random sequence. Each call produces 8 of the 32 inner mask bits.

Listing D-1 Pseudo-random number generator

```
int IDI_C2( int M )
{
    int S = 0x0000;
```

```
for ( int i = 0x0080; i > 0; i >>= 1 )
{
    if ( ( i & M ) != 0 )
    {
        S = ((( S << 1 ) ^ 0x0100 ) & 0x01FF );
    }
    else
    {
        S = (( S << 1 ) & 0x01FF );
    }

    if ( ( S & 0x0100 ) != 0 )
    {
        S = (( S ^ 0x012F ) & 0x01FF );
    }
}

return S;
}
```

A reference implementation of the decryption process, in generic source code, is available from Apple upon request.

HD Radio Tagging

In addition to their audio content, HD radio broadcasts contain Program Service Data (PSD) and Station Information Service (SIS) data. The HD radio accessory must capture these data streams and write their information to the plist fields listed in [Table D-14](#) (page 560). The accessory must populate every plist field for which it receives data from the radio broadcast, including program service data (PSD) and station information services (SIS).

Table D-14 Required plist fields for HD radio

Name	Type	Description	HD Radio Source
Name	string	The name of the track.	HD PSD
Artist	string	The name of the artist.	HD PSD
Album	string	The name of the album.	HD PSD
Genre	string	The genre classification of the track as communicated through the HD program type. The receiver must write this field as received, without decoding. It can contain ID3 codes, 2-digit genre strings, or both.	HD PSD
iTunesSongID	integer	The iTunes song identifier.	HD PSD
iTunesStorefrontID	integer	The iTunes storefront identifier.	HD PSD

Name	Type	Description	HD Radio Source
StationFrequency	string	The station's frequency, expressed only by digits and an optional decimal point.	HD SIS
StationCallLetters	string	The call letters for the station, written exactly as received. Call letters may be up to 12 characters long if the Universal Short Station Name is broadcast.	HD SIS
TimeStamp	date	The date and time at which the tagged track was broadcast, in ASCII timestamp format (e.g., 2008-04-16T00:35:42Z). The time zone must be UTC; iTunes converts it into local time. The TimeStamp plist field should be written only if the receiver has received valid, GPS-locked time information over the air and has created a valid timestamp based on it. Receivers using TI DRI352 baseband processor firmware prior to version 17 or NXP SAF355X baseband processor firmware prior to Version 2.1.2.1 should never write this field, because only newer baseband processor versions process the timestamp information correctly. Contact iBiquity for further details.	HD SIS
PodcastFeedURL	integer	Indication that the broadcast has an associated podcast. Set to 1 if there is a Podcast URL stored in the StationURL field.	HD PSD
StationURL	string	The URL of the broadcast station.	HD PSD
ProgramNumber	integer	The station's multicast channel in the range 1 to 8, where 1 indicates the main program.	HD HOST
iTunesAffiliateID	string	The iTunes affiliate ID, as broadcast.	HD PSD
UnknownData	data	A field used to pass unknown UFID data to iTunes. See "Handling Unknown Metadata Types" (page 539).	HD PSD (UFID data)

Satellite Radio Tagging

In addition to their audio content, Satellite radio broadcasts contain Service Element data. The Satellite radio accessory must capture these data streams and write their information to the plist fields listed in [Table D-15](#) (page 561). The accessory must populate every plist field for which it receives data from the radio broadcast.

Table D-15 Required plist fields for Satellite radio

Name	Type	Description	Satellite Radio Source
Name	string	The name of the track.	Sirius/XM Service Element: TrackName

Name	Type	Description	Satellite Radio Source
Artist	string	The name of the artist.	Sirius/XM Service Element: ArtistName/ArtistLabel
iTunesSongID	integer	The iTunes song identifier.	Sirius/XM Service Element: iTunesSongID
StationFrequency	string	The station's channel number.	Sirius/XM Service Element: ChannelNumber
StationCallLetters	string	The name of the broadcast station.	Sirius/XM Service Element: ChannelName
StationURL	string	The URL of the broadcast station's Web site.	Sirius/XM Service Element: URL
TimeStamp	date	The date and time at which the tagged track was broadcast; iTunes converts it into local time.	Sirius/XM Service Element: Date/Time
iTunesAffiliateID	string	The iTunes affiliate ID, as broadcast.	As assigned to Satellite Radio Receiver

Other Broadcast Tagging

In addition to their audio content, some broadcasts contain metadata that describes the currently playing broadcast audio. In these cases the radio accessory may capture these data streams and write their information to the plist fields listed in [Table D-3](#) (page 541). The accessory must populate every plist field for which it receives data from the broadcast.

Sample Tag Files

This section lists sample plist files that might be generated by FM, HD, and satellite radio tagging.

FM Radio Sample

The following is a sample FM radio plist-formatted tag file showing a header and one stored tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc./DTD PLIST 1.0/EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>MajorVersion</key>
  <integer>1</integer>
  <key>MinorVersion</key>
  <integer>1</integer>
  <key>ManufacturerID</key>
  <integer>17</integer>
```

```

<key>ManufacturerName</key>
<string>Acme</string>
<key>DeviceName</key>
<string>AC-HDR100</string>
<key>MarkedTracks</key>
<array>
  <dict>
    <key>Name</key>
    <string>Times Like These</string>
    <key>Artist</key>
    <string>Foo Fighters</string>
    <key>Album</key>
    <string>One by One</string>
    <key>iTunesSongID</key>
    <integer>6906304</integer>
    <key>iTunesStorefrontID</key>
    <integer>143441</integer>
    <key>StationFrequency</key>
    <string>104.9</string>
    <key>StationCallLetters</key>
    <string>KCNL</string>
    <key>StationURL</key>
    <string>http://www.channel1049.com</string>
    <key>Genre</key>
    <string>(20) Alternative</string>
    <key>TimeStamp</key>
    <date>2007-04-16T00:35:42Z</date>
    <key>ProgramNumber</key>
    <integer>1</integer>
    <key>iTunesStationID</key>
    <string>10491557</string>
    <key>UnknownData</key>
    <data>MDcwNDEyMzQwMTAyMTEwNDA4MTJhYjU2Z2g=</data>
  </dict>
</array>
</dict>
</plist>

```

HD Radio Samples

The following is a sample HD radio plist-formatted tag file showing a header and one stored tag:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>MajorVersion</key>
  <integer>1</integer>
  <key>MinorVersion</key>
  <integer>1</integer>
  <key>ManufacturerID</key>
  <integer>17</integer>
  <key>ManufacturerName</key>
  <string>Acme</string>
  <key>DeviceName</key>

```

```

<string>AC-HDR100</string>
<key>MarkedTracks</key>
<array>
  <dict>
    <key>Name</key>
    <string>Times Like These</string>
    <key>Artist</key>
    <string>Foo Fighters</string>
    <key>Album</key>
    <string>One by One</string>
    <key>iTunesSongID</key>
    <integer>6906304</integer>
    <key>iTunesStorefrontID</key>
    <integer>143441</integer>
    <key>StationFrequency</key>
    <string>104.9</string>
    <key>StationCallLetters</key>
    <string>KCNL</string>
    <key>PodcastFeedURL</key>
    <integer>0</integer>
    <key>StationURL</key>
    <string>http://www.channel1049.com</string>
    <key>Genre</key>
    <string>(20) Alternative</string>
    <key>TimeStamp</key>
    <date>2007-04-16T00:35:42Z</date>
    <key>ProgramNumber</key>
    <integer>1</integer>
    <key>iTunesAffiliateID</key>
    <string>00-12uR34oIcpQ</string>
    <key>UnknownData</key>
    <data>MDcwNDEyMzQwMTAyMTEwNDA4MTJhYjU2Z2g=</data>
  </dict>
</array>
</dict>
</plist>

```

The following is a sample HD radio plist-formatted tag file showing how ambiguous tags are saved to an Apple device:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc./DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>MajorVersion</key>
  <integer>1</integer>
  <key>MinorVersion</key>
  <integer>1</integer>
  <key>ManufacturerID</key>
  <integer>17</integer>
  <key>ManufacturerName</key>
  <string>Acme</string>
  <key>DeviceName</key>
  <string>AC-HDR100</string>
  <key>MarkedTracks</key>
  <array>
    <dict>

```

```

<key>AmbiguousTag</key>
<integer>1</integer>
<key>ButtonPressed</key>
<integer>1</integer>
<key>Name</key>
<string>Times Like These</string>
<key>Artist</key>
<string>Foo Fighters</string>
<key>Album</key>
<string>One by One</string>
<key>iTunesSongID</key>
<integer>6906304</integer>
<key>iTunesStorefrontID</key>
<integer>143441</integer>
<key>StationFrequency</key>
<string>104.9</string>
<key>StationCallLetters</key>
<string>KCNL</string>
<key>PodcastFeedURL</key>
<integer>0</integer>
<key>StationURL</key>
<string>http://www.channel1049.com</string>
<key>Genre</key>
<string>(20) Alternative</string>
<key>TimeStamp</key>
<date>2007-04-16T00:35:42Z</date>
<key>ProgramNumber</key>
<integer>1</integer>
<key>iTunesAffiliateID</key>
<string>00-12uR34oIcpQ</string>
</dict>
<dict>
  <key>AmbiguousTag</key>
  <integer>1</integer>
  <key>ButtonPressed</key>
  <integer>0</integer>
  <key>Name</key>
  <string>Vertigo</string>
  <key>Artist</key>
  <string>U2</string>
  <key>Album</key>
  <string>How to Dismantle an Atomic Bomb</string>
  <key>iTunesSongID</key>
  <integer>29600235</integer>
  <key>iTunesStorefrontID</key>
  <integer>143441</integer>
  <key>StationFrequency</key>
  <string>104.9</string>
  <key>StationCallLetters</key>
  <string>KCNL</string>
  <key>PodcastFeedURL</key>
  <integer>0</integer>
  <key>StationURL</key>
  <string>http://www.channel1049.com</string>
  <key>Genre</key>
  <string>(17) Rock</string>
  <key>TimeStamp</key>
  <date>2007-04-16T00:35:45Z</date>

```

iTunes Tagging

```

        <key>ProgramNumber</key>
        <integer>1</integer>
        <key>iTunesAffiliateID</key>
        <string>00-12uR34oIcpQ</string>
    </dict>
</array>
</dict>
</plist>

```

The following is a sample HD radio plist-formatted tag file showing how an accessory that has multiple tags stored locally writes those tags to an Apple device:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>MajorVersion</key>
    <integer>1</integer>
    <key>MinorVersion</key>
    <integer>1</integer>
    <key>ManufacturerID</key>
    <integer>17</integer>
    <key>ManufacturerName</key>
    <string>Acme</string>
    <key>DeviceName</key>
    <string>AC-HDR100</string>
    <key>MarkedTracks</key>
    <array>
        <dict>
            <key>Name</key>
            <string>Times Like These</string>
            <key>Artist</key>
            <string>Foo Fighters</string>
            <key>Album</key>
            <string>One by One</string>
            <key>iTunesSongID</key>
            <integer>6906304</integer>
            <key>iTunesStorefrontID</key>
            <integer>143441</integer>
            <key>StationFrequency</key>
            <string>104.9</string>
            <key>StationCallLetters</key>
            <string>KCNL</string>
            <key>PodcastFeedURL</key>
            <integer>0</integer>
            <key>StationURL</key>
            <string>http://www.channel1049.com</string>
            <key>Genre</key>
            <string>(20) Alternative</string>
            <key>TimeStamp</key>
            <date>2007-04-16T00:35:42Z</date>
            <key>ProgramNumber</key>
            <integer>1</integer>
            <key>iTunesAffiliateID</key>
            <string>00-12uR34oIcpQ</string>
        </dict>
    </dict>
</plist>

```



```

    <key>Name</key>
    <string>Vertigo</string>
    <key>Artist</key>
    <string>U2</string>
    <key>Album</key>
    <string>How to Dismantle an Atomic Bomb</string>
    <key>iTunesSongID</key>
    <integer>29600235</integer>
    <key>iTunesStorefrontID</key>
    <integer>143441</integer>
    <key>StationFrequency</key>
    <string>104.9</string>
    <key>StationCallLetters</key>
    <string>KCNL</string>
    <key>PodcastFeedURL</key>
    <integer>0</integer>
    <key>StationURL</key>
    <string>http://www.channel1049.com</string>
    <key>Genre</key>
    <string>(17) Rock</string>
    <key>TimeStamp</key>
    <date>2007-04-16T00:43:45Z</date>
    <integer>1</integer>
    <key>ProgramNumber</key>
    <integer>1</integer>
    <key>iTunesAffiliateID</key>
    <string>00-12uR34oIcpQ</string>
  </dict>
  <dict>
    <key>Name</key>
    <string>Ball and Chain</string>
    <key>Artist</key>
    <string>Social Distortion</string>
    <key>Album</key>
    <string>Social Distortion</string>
    <key>iTunesSongID</key>
    <integer>197986000</integer>
    <key>iTunesStorefrontID</key>
    <integer>143441</integer>
    <key>StationFrequency</key>
    <string>104.9</string>
    <key>StationCallLetters</key>
    <string>KCNL</string>
    <key>PodcastFeedURL</key>
    <integer>0</integer>
    <key>StationURL</key>
    <string>http://www.channel1049.com</string>
    <key>Genre</key>
    <string>(17) Rock</string>
    <key>TimeStamp</key>
    <date>2007-04-16T00:48:45Z</date>
    <key>ProgramNumber</key>
    <integer>1</integer>
    <key>iTunesAffiliateID</key>
    <string>00-12uR34oIcpQ</string>
  </dict>
</array>
</dict>

```

```
</plist>
```

Satellite Radio Sample

The following is a sample satellite radio plist-formatted tag file showing a header and one stored tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>MajorVersion</key>
  <integer>1</integer>
  <key>MinorVersion</key>
  <integer>1</integer>
  <key>ManufacturerID</key>
  <integer>17</integer>
  <key>ManufacturerName</key>
  <string>Acme</string>
  <key>DeviceName</key>
  <string>AC-HDR100</string>
  <key>MarkedTracks</key>
  <array>
    <dict>
      <key>Name</key>
      <string>Times Like These</string>
      <key>Artist</key>
      <string>Foo Fighters</string>
      <key>iTunesSongID</key>
      <integer>6906304</integer>
      <key>StationFrequency</key>
      <string>9</string>
      <key>StationCallLetters</key>
      <string>90's on 9</string>
      <key>Genre</key>
      <string>(20) Alternative</string>
      <key>TimeStamp</key>
      <date>2007-04-16T00:35:42Z</date>
      <key>iTunesAffiliateID</key>
      <string>00-12uR34oIcpQ</string>
      <key>UnknownData</key>
      <data>MDcwNDEyMzQwMTAyMTwNDA4MTJhYjU2Z2g=</data>
    </dict>
  </array>
</dict>
</plist>
```

Nike + iPod Cardio Equipment System

The Nike + iPod cardio equipment feature lets the user of an Apple device record workout data gathered by an attached sports or exercise device, while at the same time enjoying entertainment from the Apple device.

An Apple device can support only one such accessory at any time, and that accessory must be attached using the 30-pin connector. Currently the 2G touch, the 3G nano, and the 4G nano support this system.

Using Nike + iPod Cardio Equipment

To use the capabilities of the Nike + iPod cardio equipment feature, an Apple device user typically does the following:

1. The user connects a compatible Apple device to a compatible piece of cardio equipment, makes an entertainment selection, and selects either QuickStart or a specific workout type.
2. The cardio equipment downloads user preferences and other information from the Apple device.
3. If there is enough space on the Apple device to store the workout data, the cardio equipment displays the message “Recording workout to iPod.” It opens a file and records data in real time.
4. The cardio equipment prompts the user to accept the weight setting retrieved from the Apple device or change it, in which case it stores the new weight on the Apple device.
5. When the workout ends, the cardio equipment closes the file on the Apple device and displays a closing message with a workout summary. It confirms that the workout has been recorded and tells the user how to view the results.
6. The user connects the Apple device to a computer and iTunes uploads the workout files to nikeplus.com, where the user can view the results.

Cardio Equipment Design

This section specifies the requirements for cardio equipment that works with the Nike + iPod system.

Determining Apple Device Support for Cardio Equipment

To use the Nike + iPod feature, the cardio equipment must first determine that the connected Apple device supports the required minimum iOs versions shown in [Table E-1](#) (page 570).

Table E-1 Lingo version support

Lingo	Minimum Required Protocol Version
0x09: Sports	v1.01
0x0C: Storage	v1.02

If the minimum required protocol versions are supported by the connected Apple device, the cardio equipment must next identify itself as supporting those lingo; see [“Identification Procedure”](#) (page 570).

The final step is for the cardio equipment to query the Sports lingo capabilities of the Apple device with a `GetiPodOptionsForLingo` command and then parse the resulting `RetiPodOptionsForLingo` command for the Nike + iPod Cardio Equipment bit (bit 1); see [“Command 0x4C: RetiPodOptionsForLingo”](#) (page 166).

Note: All Sports lingo commands require authentication. The cardio equipment must not send the `GetiPodOptionsForLingo` command until after the authentication process has begun. See [“Apple Device Authentication of the Accessory”](#) (page 73) for further information.

Identification Procedure

The cardio equipment must perform an identification procedure by which it discovers the capabilities of the connected Apple device. If it attempts to identify for a lingo not supported by the attached Apple device, the identification fails and the Apple device displays an error message.

The following identification procedure is recommended:

1. Send a `StartIDPS` command. If the Apple device responds with an `ACK` command passing 0x04 (Bad Parameter), the accessory must send `IdentifyDeviceLingoes` instead, with `deviceId=0x0000` and identifying only the General lingo with no options.
2. Query the Apple device for the version numbers of all necessary lingo.
3. Based on the responses to the query, form appropriate FID values, declare them for the required lingo, and request authentication if necessary.
4. End the IDPS process by sending `EndIDPS`.

Sample Identification Processes

This section presents two sample identification sequences for a cardio equipment accessory. In both cases the sequences begin with the accessory sending a `StartIDPS` command. However, in both cases the Apple device does not support IDPS, so the accessory reverts to sending `IdentifyDeviceLingoes`. For examples of identification sequences in which IDPS is supported, see [“Sample Identification Sequences”](#) (page 402).

For a first example, consider a piece of cardio equipment that is intended to offer simple Apple device playback control (play/pause, next/previous track, etc.), display information about the currently playing track, and log workout data to the Apple device. To achieve these goals, the cardio equipment needs access to four iAP lingo:

- Simple Remote
- Display Remote
- Sports
- Storage

The example in [Table E-2](#) (page 571) shows the command traffic exchanged between the cardio equipment and an iPod Classic running software v1.1.1.

Table E-2 Sample identification process 1

cardio equipment	iPod Classic (v1.1.1)
StartIDPS	
	ACK(StartIDPS, 0x04 = Bad Parameter)
IdentifyDeviceLingoes (Lingoes: 0x0001, Options: 0x00, DeviceID: 0x0000)	
	ACK(IdentifyDeviceLingoes, Success)
RequestLingoProtocolVersion(0x02)	
	ReturnLingoProtocolVersion(0x02, v1.02)
RequestLingoProtocolVersion(0x03)	
	ReturnLingoProtocolVersion(0x03, v1.05)
RequestLingoProtocolVersion(0x09)	
	ACK(RequestLingoProtocolVersion, Bad Parameter) (This Apple device does not support the Sports lingo.)
RequestLingoProtocolVersion (0x0C)	
	ReturnLingoProtocolVersion(0x0C, v1.01) (This Apple device does not support the required version of the Storage lingo.)
IdentifyDeviceLingoes(Lingoes:0x000D, Options:0x02, DeviceID:0x0200)	
	ACK(IdentifyDeviceLingoes, Success)
	GetDeviceAuthenticationInfo()

In this first example, the cardio equipment determines that the attached Apple device does not support the proper versions of the Sports and Storage lingoes and thus does not attempt to identify for those lingoes. Display Remote and Simple Remote are both supported, and the cardio equipment properly identifies and authenticates for those lingoes.

A more successful example is shown in [Table E-3](#) (page 572). It lists the command traffic exchanged between the cardio equipment and a 3G iPod nano updated to support the Nike + iPod feature.

Table E-3 Sample identification process 2

Cardio equipment	3G iPod nano (version 1.1.2)
StartIDPS	
	ACK(StartIDPS, 0x04 = Bad Parameter)
IdentifyDeviceLingoes (Lingoes:0x0001, Options:0x00, DeviceID:0x0000)	
	ACK(IdentifyDeviceLingoes, Success)
RequestLingoProtocolVersion(0x02)	
	ReturnLingoProtocolVersion(0x02, v1.02)
RequestLingoProtocolVersion(0x03)	
	ReturnLingoProtocolVersion(0x03, v1.05)
RequestLingoProtocolVersion(0x09)	
	ReturnLingoProtocolVersion(0x09, v1.01)
RequestLingoProtocolVersion(0x0C)	
	ReturnLingoProtocolVersion(0x0C, v1.02)
IdentifyDeviceLingoes(Lingoes:0x120D, Options:0x02, DeviceID: 0x0200)	
	ACK(IdentifyDeviceLingoes, Success)
	GetDeviceAuthenticationInfo()

In this second example, the Apple device supports the proper minimum protocol versions for each of the necessary lingoes. The cardio equipment identifies for all of those lingoes and operation proceeds normally.

Declaring Cardio Equipment Support to the Apple Device

In the future, Apple devices may change their user interface elements when attached to Nike + iPod cardio equipment. Hence, the cardio equipment must declare its support of the Nike + iPod feature. This is also necessary so that the Apple device can differentiate between types of Sports lingo accessories (e.g. between a Nike + iPod Sports Kit Receiver and a stair climber).

The cardio equipment declares itself as such by supporting the Sports lingo `GetDeviceCaps` command and replying to it with a `RetDeviceCaps` command with `capsMask` bit 9 = 1, indicating to the Apple device that the cardio equipment supports the required commands.

Accessing User Data

Nike + iPod compatible cardio equipment can access certain user data stored on an Apple device. See [“Sports Lingo commands”](#) (page 336) for detailed information on the commands available for retrieving and/or setting the available user data fields.

Proper use of this user data requires that the cardio equipment must:

1. Always get/set data for the current `userIndex`. It is assumed that the Apple device is already configured for the correct `userIndex`.
2. Write `userData` fields back to the Apple device if, and only if, the user entered new or updated information via the cardio equipment user interface. For example, if the cardio equipment uses a default value for weight during a workout, that weight must not be written back to the `userData` on the Apple device.
3. Never display the name field for `userIndex = 0`. That value is reserved for the default new or unknown user; consequently, the associated name field is invalid. The `userData` name fields for other `userIndex` values should be valid.

Recording Workout Data

Workout data is stored on the Apple device as a UTF-8 encoded, XML-formatted file. The cardio equipment is responsible for writing the data directly to the Apple device's file system via the Storage lingo. Appropriate XML elements are written to the file in real time as the workout proceeds.

Writing Workout Data

After successfully providing valid accessory authentication information to the attached Apple device, the cardio equipment typically uses the following command sequence to write the workout data file to the Apple device. For details of these commands, see [“Lingo 0x09: Sports Lingo”](#) (page 336) and [“Lingo 0x0C: Storage Lingo”](#) (page 362).

Table E-4 Writing workout data to the Apple device

Cardio equipment	Apple device (with Nike + iPod feature support)
SportsLingo: GetiPodCaps()	
	SportsLingo: RetiPodCaps()
StorageLingo: GetiPodCaps()	
	StorageLingo: RetiPodCaps()
OpeniPodFeatureFile(featureType:0x02, optionsMask:0x000B, fileData:"</gymData>")	
	RetiPodFileHandle()

Cardio equipment	Apple device (with Nike + iPod feature support)
<code>WriteiPodFileData()</code> as many times as needed	
	<code>iPodACK(WriteiPodFileData, status)</code> for each <code>WriteiPodFileData</code> command
<code>CloseiPodFile()</code>	
	<code>iPodACK(CloseiPodFile, ackStatus)</code>

The cardio equipment starts by sending a `GetiPodCaps` command to retrieve the Sports lingo capabilities of the Apple device. The Apple device responds with a `RetiPodCaps` command.

If the Apple device indicates that it supports the Nike + iPod feature, the cardio equipment next sends a `GetiPodCaps` command to determine the Storage lingo capabilities of the Apple device. The Apple device responds with a `RetiPodCaps` command containing the following data:

- `totalSpace`: the amount of storage on the Apple device, including space currently in use
- `maxFileSize`: the largest possible size of a file on the Apple device
- `maxWriteSize`: the largest amount of data that can be written to the Apple device in a single `WriteiPodFileData` command
- `majorVersion` and `minorVersion`: the version number of the Storage lingo protocol (currently 1.02)

The cardio equipment is required to honor the `maxWriteSize` limitation of the Apple device; it must never send a `WriteiPodFileData` command with a data payload larger than that `maxWriteSize` value.

To create a cardio workout file on the Apple device, the cardio equipment sends an `OpeniPodFeatureFile` command with `feature = 0x02`, `options = 0x000B`, and `fileData` equal to the UTF-8 representation of the string necessary to close the root element: `"</gymData>"` or `0x3C 0x2F 0x67 0x79 0x6D 0x44 0x61 0x72 0x61 0x3E`. On receiving this command, the Apple device performs the following steps:

1. Creates a new workout file in `/iPod_Control/Device/Trainer/Workouts/Empeds/USERNAME/latest/` (where `USERNAME` is the name of the current `userIndex`)
2. Sends a `RetiPodFileHandle` command with a handle to the new file

Next, the cardio equipment must use a series of `WriteiPodFileData` commands to write the following elements to the file:

the XML version header line
 root element opening tag: `<gymData>`
`<vers>`
`<equipmentInfo>`
`<userInfo>`
`<template>` (optional)
`<interval>` with `<event>` "start" or "continue" and initial values for all child elements

See ["XML Element Formats"](#) (page 576) for details about valid XML elements.

As the workout progresses, the cardio equipment is required to write `<interval>` XML elements on a real-time basis at least every 10 seconds. Certain child elements of `<interval>` (the `<incline>` element, for example) are required only if their value has changed since the previous `<interval>` element.

Note: All `<interval>` elements written to file must be complete and closed. Because a user can detach the Apple device at any time during the process, `<interval>` elements must not be broken across multiple `WriteiPodFileData` commands.

At the end of the workout, the cardio equipment must write the `<workoutSummary>` element to the file and then send the `CloseiPodFile` command. Upon receiving the `CloseiPodFile` command, the Apple device performs the following actions (assuming the proper option bits were set in `OpeniPodFeatureFile`):

1. Append the `<ipodInfo>` XML element to file.
2. Write the specified `fileData` to close the root element.
3. Close the XML file.
4. Compute and insert an XML `<Signature>` element in to file.
5. Send an `iPodACK` command acknowledging success.

While performing these steps, the Apple device is unresponsive to further iAP commands. Signature calculation requires many seconds for long workouts. The cardio equipment must wait for the `CloseiPodFile` command to fully complete before sending any new iAP commands. The `CloseiPodFile` command is complete when the Apple device returns an `iPodACK` with any status other than command pending (0x06).

Workout Events

Events that occur as part of a normal workout must be tracked and recorded to the XML file as part of the `<event>` child element of `<interval>`. There are five event types to be recorded:

- **start:** the first `<interval>` of a new workout
- **continue:** the first `<interval>` of a workout already in progress
- **pause:** when a user pauses the active workout
- **resume:** when a user resumes the workout
- **end:** the final `<interval>` of a completed workout

Workout time must not elapse between a pause event and a resume event; the `<sec>` children of both the pause and the resume `<interval>` elements must be identical. Continue events are used to mark a file that was opened after the workout has already started; some amount of `<interval>` data from the beginning of the workout is not present in the file.

File System Full

The Apple device will determine whether or not enough disk space is available to record and sign the workout data. When the Apple device's file system no longer contains enough free space, the Apple device will respond with `iPodACK = (0x03) (out of resources)` to any `OpeniPodFeatureFile` or `WriteiPodFileData` commands. If this occurs during an `OpeniPodFeatureFile` command, the cardio equipment must notify the user that the Apple device is full and that the workout will not be recorded.

If an "out of resources" `iPodACK` is received in response to a `WriteiPodFileData`, the cardio equipment must close the current workout file without writing an end `<interval>` or `<workoutSummary>` and must alert the user that the Apple device is full and that the workout is no longer being recorded.

File Writing Error Recovery

Cardio equipment accessories must properly handle unexpected failures during the file writing process. In the event of a failure while writing data to the Apple device (as indicated by lack of an `ackStatus=Success` in response to `WriteiPodFileData`), the cardio equipment should first retry the operation. If the operation fails twice, the current workout file must be closed and the cardio equipment must attempt to open a new workout file.

Recording a Workout Already in Progress

When opening a new workout data file mid-workout, the standard procedure for creating a new workout file must be followed. The difference between a standard workout file and one started in the middle of a workout is that the first `<interval>` element will have a nonzero `<sec>` value and an `<event>` continue (instead of start) value. It is the responsibility of the end consumer of the data (such as `nikeplus.com`) to recognize a workout file that started mid-workout and act appropriately.

XML Element Formats

All file data must be well-formed and valid XML. Character entities are not currently supported by the Apple device. All special characters (`&`, `<`, `>`) used as element data content must be encapsulated as part of a CDATA section. For human readability, it is recommended that each element close tag be followed by a newline character (`0x0A`). XML elements are required to appear in the order shown in [Table E-5](#) (page 577) with two exceptions:

- The `<userInfo>` element may appear at any point in the file.
- The `<template>` element may appear multiple times and at any point in the file.

All `<interval>` elements must be written in chronological order (ascending `<sec>` values). All distances must be recorded to 0.01 km resolution. All kilocalorie values must be recorded to 0.01 kCal resolution. In situations where the user is not moving and the workout speed is essentially zero, the `<pace>` element should be set to 0 instead of to infinity.

Table E-5 XML element usage

Element	Format	Example	Treadmill	Elliptical	Stepper	Bike	Other
Header							
XML Header		<?xml version="1.0" encoding="UTF-8"?>	Req	Req	Req	Req	Req
Root Element Open		<gymData>	Req	Req	Req	Req	Req
File Format Version	integer	<vers>1</vers>	Req	Req	Req	Req	Req
Equipment Info							
		<equipmentInfo>	Req	Req	Req	Req	Req
Manufacturer ID	integer converted to hexadecimal	<manufacturerID>A5C57</manufacturerID> (integer = 678999; see Note below)	Req	Req	Req	Req	Req
Manufacturer Name	string	<manufacturerName>Acme</manufacturerName>	Req	Req	Req	Req	Req
Equipment Type	string enum: Treadmill, Elliptical, Stepper, Bike, or Other	<type>Treadmill</type>	Req	Req	Req	Req	Req
Equipment Model	string	<model> ExerPro 5000</model>	Req	Req	Req	Req	Req
Equipment Serial No.	string	<serialNumber>XYZ012345TG88</serialNumber>	Opt	Opt	Opt	Opt	Opt
Gym Name	string	<gymName>GloboGym</gymName>	Opt	Opt	Opt	Opt	Opt
Gym Location	string	<gymLocation>Cupertino, CA</gymLocation>	Opt	Opt	Opt	Opt	Opt
		</equipmentInfo>	Req	Req	Req	Req	Req
User Info							

Element	Format	Example	Tread-mill	Elliptical	Stepper	Bike	Other
		<userInfo>	Req	Req	Req	Req	Req
User Name	string	<name>Jane Doe</name>	Opt	Opt	Opt	Opt	Opt
Weight	float kilograms	<kg>53.5</kg>	Req	Req	Req	Opt	Opt
Gender	string enum: female or male	<gender>female</gender>	Opt	Opt	Opt	Opt	Opt
		</userInfo>	Req	Req	Req	Opt	Opt
Workout Template							
		<template>	Opt	Opt	Opt	Opt	Opt
Template Name	string	<templateName>Hills</templateName>	Opt	Opt	Opt	Opt	Opt
Caloric Goal	float kilocalories	<kCal>200.00</kCal>	Opt	Opt	Opt	Opt	Opt
Time Goal	integer seconds	<sec>900</sec>	Opt	Opt	Opt	Opt	Opt
Distance Goal	float kilometers	<km>10.00</km>	Opt	Opt	N/A	Opt	Opt
	float floors	<floors>30.0</floors>	N/A	N/A	Opt	N/A	Opt
Speed Goal	integer seconds per kilometer	<pace>450</pace>	Opt	N/A	N/A	N/A	Opt
	integer steps/strides per minute	<spm>40</spm>	N/A	Opt	Opt	N/A	Opt
	integer revolutions per minute	<rpm>60</rpm>	N/A	N/A	N/A	Opt	Opt
Heart Rate Goal	integer beats per minute	<bpm>140</bpm>	Opt	Opt	N/A	Opt	Opt
		</template>	Opt	Opt	Opt	Opt	Opt
Interval Data (written at least every 10 seconds)							
		<interval>	Req	Req	Req	Req	Req
Event Type	string enum: start, continue, pause, resume, or end	<event>pause</event>	Opt	Opt	Opt	Opt	Opt

Element	Format	Example	Tread-mill	Ellip-tical	Stepper	Bike	Other
Current Elapsed Time	integer seconds	<sec>1200</sec>	Req	Req	Req	Req	Req
Current Calories	float kilocalories	<kCal>230.12</kCal>	Req	Req	Req	Req	Req
Current Distance	float kilometers	<km>3.25</km>	Req	Req	N/A	Req	Opt
	float floors	<floors>11.0</floors>	N/A	N/A	Req	N/A	Opt
Current Speed	integer seconds per kilometer	<pace>430</pace>	Req	N/A	N/A	N/A	Opt
	integer steps/strides per minute	<spm>40</spm>	N/A	Req	Req	N/A	Opt
	integer revolutions per minute	<rpm>55</rpm>	N/A	N/A	N/A	Req	Opt
Current Heart Rate	integer beats per minute	<bpm>101</bpm>	Opt	Opt	Opt	Opt	Opt
Incline	float percent	<incline>10.1</incline>	Required only if value has changed				
Resistance/Effort	integer level	<level>3</level>	Required only if value has changed				
		</interval>	Req	Req	Req	Req	Req
Workout Summary							
		<workoutSummary>	Req	Req	Req	Req	Req
Total Elapsed Time	integer seconds	<sec>2450</sec>	Req	Req	Req	Req	Req
Total Calories	float kilocalories	<kCal>342.34</kCal>	Req	Req	Req	Req	Req
Total Distance	float kilometers	<km>10.15</km>	Req	Req	N/A	Req	Opt
	float floors	<floors>11.0</floors>	N/A	N/A	Req	N/A	Opt
Average Speed	integer seconds per kilometer	<pace>430</pace>	Req	N/A	N/A	N/A	Opt
	integer steps/strides per minute	<spm>40</spm>	N/A	Req	Req	N/A	Opt

Element	Format	Example	Tread-mill	Ellip-tical	Stepper	Bike	Other
	integer revolutions per minute	<rpm>55</rpm>	N/A	N/A	N/A	Req	Opt
Average Heart Rate	integer beats per minute	<bpm>140</bpm>	Opt	Opt	Opt	Opt	Opt
		</workoutSummary>	Req	Req	Req	Req	Req
Apple Device Info							
		<ipodInfo>	Written by Apple device				
File Open Date/Time	date/time format w/ offset	<openTime> 2007-12-17T 17:15:17-08:00 </openTime>	Written by Apple device				
File Close Date/Time	date/time format w/ offset	<closeTime> 2007-12-17T 17:56:07-08:00 </closeTime>	Written by Apple device				
Apple device Model	string	<model>MA980</model>	Written by Apple device				
Apple Device Software Version	string	<softwareVersion> 1.1.2</softwareVersion>	Written by Apple device				
Apple device Serial No.	string	<serialNumber> 4H802998TVS </serialNumber>	Written by Apple device				
		</ipodInfo>	Written by Apple device				
XML Signature							
Apple device signature		<Signature> ...</Signature>	Written by Apple device				
Root Close							
		</gymData>	Written by Apple device				

Note: The Manufacturer ID element is the hexadecimal equivalent of the first part of a product ID assigned by Apple, which identifies the licensee's account. For example, if Apple assigns a product ID of 678999-12345, the Manufacturer ID element must contain the hexadecimal equivalent of 678999, which is A5C57. To obtain a product ID number, the licensee must submit a product plan to Apple's MFi portal.

User Interface Requirements

This section sets forth the requirements for the cardio equipment's user interface. An example of a typical user interface process, from the perspective of the cardio equipment, is shown in [Figure E-1](#) (page 583).

The Apple Device Does Not Support Nike + iPod

The cardio equipment must determine whether the attached Apple device supports the Nike + iPod feature; see ["Determining Apple Device Support for Cardio Equipment"](#) (page 569). If the attached Apple device does not support the Nike + iPod feature, then the cardio equipment must display this information to the user. See [Table E-6](#) (page 582) for the approved UI string.

Deciding to Record a Workout to the Apple Device

The cardio equipment must first check the Apple device for the user's Workout Recording Preference, using the Sports lingo. If the user's recording preference is set to Never then the workout must not be recorded. If the user's preference is set to Always, Ask, or Not Set, then the cardio equipment must present an option to the user. There are two approved forms for this option:

- Recommended: allow user to opt out. Default to recording the workout, but present an option for the user to cancel the recording.
- Allowed: present a dialog prompting the user to choose whether or not they would like the workout to be recorded.

The Apple Device is Full

In the event that the cardio equipment is unable to record a workout to the Apple device because of insufficient disk space, the cardio equipment must present a message to the user alerting them to this problem. See [Table E-6](#) (page 582) for the approved UI string. Note that this problem may occur either when trying to open a workout file or mid-workout, if the Apple device's file system becomes full.

User Weight

Workouts are displayed at nikeplus.com using a metric known as Cardio Miles. Cardio Miles are a conversion from workout calories to equivalent running miles. The key metric recorded in most cardio equipment workouts is the calories burned. To accurately calculate calories burned, the user must be required to enter a weight value.

Because a user's weight may change over time, the equipment must always present the user with an opportunity to enter or adjust the weight value. The enter-weight UI on the cardio equipment must start with the weight value retrieved from Apple device via the `GetUserData` command. It is acceptable for the cardio equipment UI to automatically accept the displayed weight if the user fails to enter or adjust the weight after a length of time has passed.

If the user enters a weight different from that retrieved from the Apple device, the cardio equipment must write the new weight value back to Apple device via the `SetUserData` command.

Recording Indicator

Cardio equipment must always present a visible indicator to the user that the workout is being successfully recorded to Apple device. This indicator must be present as soon as the workout recording begins and then disappear when the workout recording ends (either because the workout is finished or the Apple device has been detached from the equipment).

Whenever the user pauses a workout, the recording indicator must flash on and off until the workout either resumes normally or is terminated.

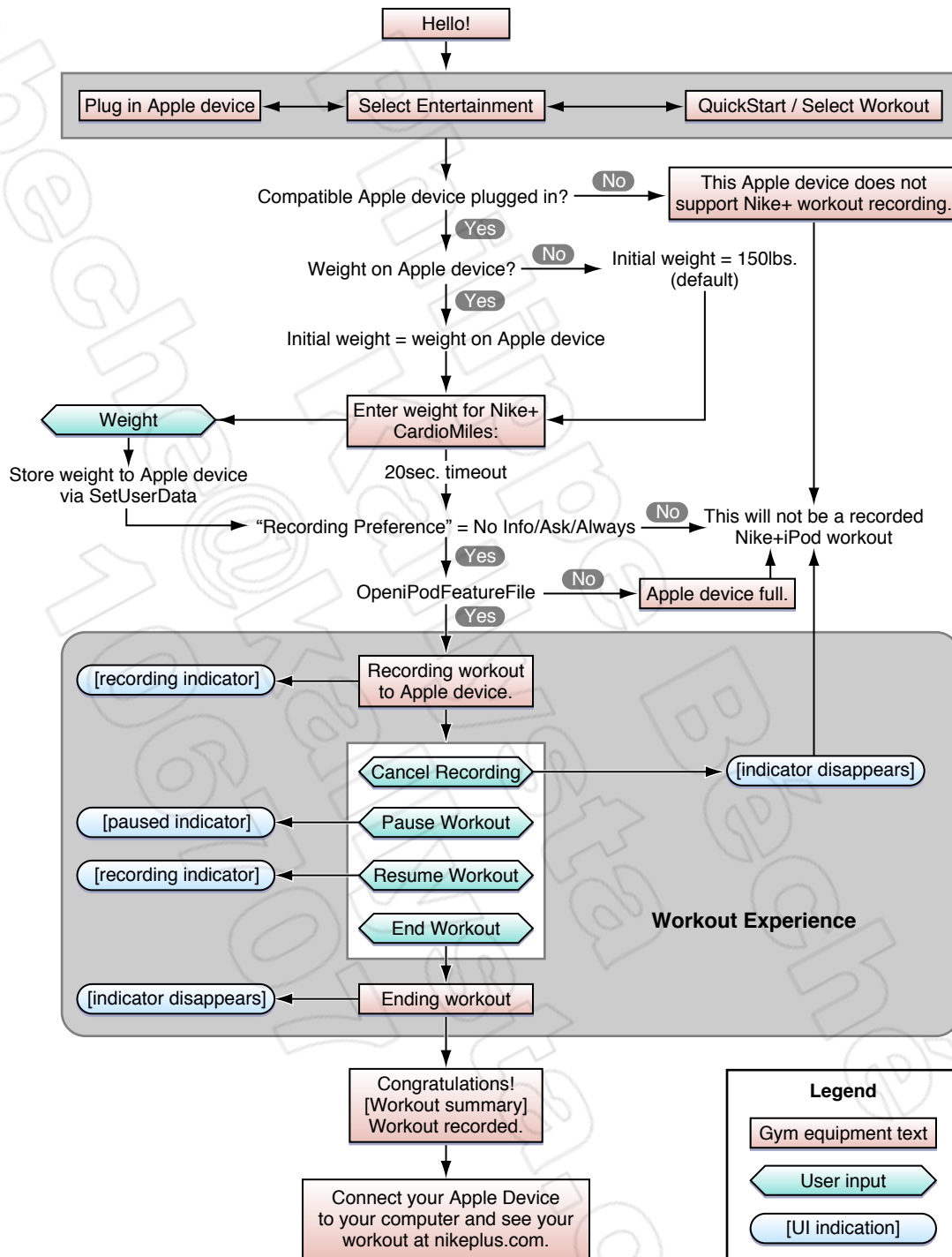
Workout Completed

At the end of a workout, the cardio equipment must present two messages to the user. The first is a congratulatory message stating that the workout has been successfully recorded and showing a summary of the workout data. The second message gives brief instruction on how to view the user's workout statistics at the nikeplus.com website. [Table E-6](#) (page 582) lists all the approved UI strings for display by Nike + iPod cardio equipment.

Table E-6 Approved user interface messages

Event	Approved UI string	Short UI string
Apple Device Does Not Support the Nike + iPod System	This Apple device does not support Nike + iPod workout recording.	This Apple device does not support workout recording.
The Apple device is full	This Apple device is full.	Apple device full.
Workout summary	Congratulations! <Workout summary> Workout recorded.	Congratulations! <Workout summary> Workout recorded.
Upload instructions	Connect your Apple device to your computer to see your workout at nikeplus.com .	See your workout at nikeplus.com .

Figure E-1 Sample user experience flow chart



Sample Command Sequence

Table E-7 (page 584) shows a sample sequence of IDPS commands that implements the Nike + iPod cardio equipment system in an accessory.

Table E-7 Cardio equipment command sequence using IDPS

Step	Accessory command	Apple device command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> ■ If the accessory receives no ACK command within 500 ms, it may retry Step 1 at 500 ms intervals as long as Apple Device Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them. ■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the Apple device does not support IDPS. Within 800 ms the accessory must send an IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). A sample command sequence is listed in Table F-47 (page 636). ■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1. 			
3	GetiPodOptions-ForLingo		getting options for General Lingo
4		RetiPodOptions-ForLingo	returning options of 000000000003F3FF (Line out usage Video output NTSC video signal format PAL video signal format Composite video out connection S-Video video out connection Component video out connection Closed captioning (video) Video aspect ratio 4:3 (fullscreen) Video aspect ratio 6:9 (widescreen) reserved app communication capable Apple device notifications) for General Lingo
5	GetiPodOptions-ForLingo		getting options for Sports Lingo
6		RetiPodOptions-ForLingo	returning options of 0000000000000002 (Nike + iPod cardio equipment) for Sports Lingo

Step	Accessory command	Apple device command	Comment
7	GetiPodOptions-ForLingo		getting options for Storage Lingo
8		RetiPodOptions-ForLingo	returning options of 0000000000000003 (iTunes tagging Nike + iPod cardio equipment) for Storage Lingo
9	SetFIDTokenValues		setting 12 FID tokens ; IdentifyToken = (lingoes: 0/9/12 options 0x00000002 device ID 0x00000200); AccCapsToken = 0x0000000000000805; AccInfoToken = Acc name (FitBicycle 2010); AccInfoToken = Acc FW version (v1.0.1); AccInfoToken = Acc HW version (v1.0.0); AccInfoToken = Acc manufacturer (Apple); AccInfoToken = Acc model number (MA1234LL/A); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'selected'); iPodPreferenceToken = (setting 'on' for preference class 'video out setting' with restore on exit 'selected'); iPodPreferenceToken = (setting 'fullscreen' for preference class 'screen configuration' with restore on exit 'selected'); iPodPreferenceToken = (setting 'NTSC' for preference class 'video format setting' with restore on exit 'selected'); iPodPreferenceToken = (setting 'component' for preference class 'video connection' with restore on exit 'selected')
10		RetFIDTokenValueACKs	12 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; iPodPreferenceToken = (line out usage) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (screen configuration) accepted; iPodPreferenceToken = (video format setting) accepted; iPodPreferenceToken = (video connection) accepted

Step	Accessory command	Apple device command	Comment
11	EndIDPS		status 'finished with IDPS; proceed to authentication'
12		IDPSStatus	status 'ready for auth'
13		GetDevAuthentication-Info	no params
14	RetDevAuthentication-Info		returning auth protocol v2.0; current section index: 0; maximum section index: 1; cert data ...
15		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
16	RetDevAuthentication-Info		returning auth protocol v2.0; current section index: 1; maximum section index: 1; cert data ...
17		AckDevAuthentication-Info	acknowledging 'auth info supported'
18		GetDeviceCaps	no params
19		GetDevAuthentication-Signature	offering challenge '...' with retry counter 1
20	RetDevAuthentication-Signature		returning signature '...'
21		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
22		GetDeviceCaps	requesting accessory Sports lingo capabilities; no params
23		GetDeviceCaps	requesting accessory Storage lingo capabilities; no params
24	RetDeviceCaps		returning 'Gym equipment command support'
25	RetDeviceCaps		returning 'file system type 'none'; lingo v1.02'
26	Get iPodCaps		requesting Apple device Sports lingo capabilities; no params
27		Ret iPodCaps	returning 'Gym equipment support User data support' with userCount of 1

Step	Accessory command	Apple device command	Comment
28	GetiPodCaps		requesting Apple device Storage lingo capabilities; no params
29		RetiPodCaps	returning 'total space 2147483648; max file size 1073741824; max write size 500; v1.02'
30	GetUserData		requesting 'Gender'
31		RetUserData	returning 'No information' for data type 'Gender'
32	GetUserData		requesting 'Age'
33		RetUserData	returning '26 years' for data type 'Age'
34	GetUserData		requesting 'Name'
35		RetUserData	returning 'NewUser' for data type 'Name'
36	GetUserData		requesting 'Weight'
37		RetUserData	returning '92.0 kg' for data type 'Weight'
38	OpeniPodFeatureFile		opening feature type 'Gym Equipment Workout' with options mask 'file data ipodInfo XML signature' and file data: </gymData>;
39		RetiPodFileHandle	returning file handle in response to command 'Storage Lingo::OpeniPodFeatureFile'
40	WriteiPodFileData		writing 39 bytes at offset 0 and returned file handle: <?xml version=""1.0"" encoding=""UTF-8""?>
41		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
42	WriteiPodFileData		writing 10 bytes at offset 39 and handle 0: <gymData>;
43		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
44	WriteiPodFileData		writing 15 bytes at offset 49 and handle 0: <vers>1</vers>;

Step	Accessory command	Apple device command	Comment
45		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
46	WriteiPodFileData		writing 166 bytes at offset 64 and handle 0: <equipmentInfo> < manufacturerID>123A3A27 </manufacturerID> < manufacturerName>Scotty </manufacturerName> < type>Bike</type> < model>Bike for Scotty</model> </equipmentInfo>
47		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
48	WriteiPodFileData		writing 36 bytes at offset 230 and handle 0: <userInfo> < kg>90</kg> </userInfo>
49		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
50	WriteiPodFileData		writing 74 bytes at offset 266 and handle 0: <template> < templateName>Goal </templateName> < kCal>500</kCal> </template>
51		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
52	WriteiPodFileData		writing 121 bytes at offset 340 and handle 0: <interval> < event>start</event> < sec>0</sec> < kCal>0</kCal> < km>0</km> < rpm>0</rpm> < bpm>0</bpm> < level>1</level> </interval>
53		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
54	WriteiPodFileData		writing 88 bytes at offset 461 and handle 0: <interval> < sec>10</sec> < kCal>0</kCal> < km>2</km> < rpm>74</rpm> < bpm>96</bpm> </interval>

Step	Accessory command	Apple device command	Comment
55		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
56	WriteiPodFileData		writing 90 bytes at offset 549 and handle 0: <interval> < sec>20</sec> < kCal>2</kCal> < km>10</km> < rpm>98</rpm> < bpm>163</bpm> </interval>
57		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
58	WriteiPodFileData		writing 126 bytes at offset 639 and handle 0: <interval> < event>end</event> < sec>30</sec> < kCal>500</kCal> < km>17</km> < rpm>87</rpm> < bpm>172</bpm> < level>2</level> </interval>
59		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
60	WriteiPodFileData		writing 104 bytes at offset 765 and handle 0: <workoutSummary> < sec>30</sec> < kCal>500</kCal> < km>17</km> < rpm>89</rpm> < bpm>167</bpm> </workoutSummary>
61		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
62	CloseiPodFile		closing file with handle 0
63		iPodAck	acknowledging 'Success' to command 'Storage Lingo::CloseiPodFile' with file handle 0

Historical Information

This appendix memorializes the specifications of past Apple device models and iAP protocols. It is included in this specification to provide guidance for developers who need to design accessories compatible with these past technologies.

Past Protocol Features

[Table F-1](#) (page 591) and [Table F-2](#) (page 593) list the past protocol versions for each lingo and the firmware releases in which they were available.

In the following tables, "NL" indicates that the given lingo was not implemented in the specified model loaded with the specified firmware. "NV" indicates that although the lingo was implemented, its protocol version could not be read from the Apple device. Footnotes are listed below each table.

Table F-1 Past Apple device models, firmware, and lingo versions, table 1

Model	Firmware		Lingoes										
	Version	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x0A	0x0C
3G ¹	2.0.0	04/03	NV	NL	NV	NL	NL	NL	NL	NL	NL	NL	NL
	2.1.0	10/03	NV	NV	NV	NL	1.00	NL	NL	NL	NL	NL	NL
	2.2.0	02/04	NV	NV	NV	NL	1.02	NL	NL	NL	NL	NL	NL
	2.3.0	02/05	NV	NV	NV	NL	1.02	NL	NL	NL	NL	NL	NL
mini	1.0.0	02/04	NV	NL	NV	NL	1.01	NL	NL	NL	NL	NL	NL
	1.1.0	03/04	NV	NL	NV	NL	1.03	NL	NL	NL	NL	NL	NL
	1.2.0	11/04	1.00	NL	1.00	NL	1.05	1.00	NL	NL	NL	NL	NL
	1.3.0	02/05	1.00	NL	1.00	NL	1.05	1.00	NL	NL	NL	NL	NL
	1.4.0	06/05	1.02	NL	1.00	1.01	1.05	1.00	NL	NL	NL	NL	NL
4G	3.0.0	07/04	NV	NV	NV	NL	1.04	NV	NL	NL	NL	NL	NL
	3.0.1	08/04	NV	NV	NV	NL	1.04	NV	NL	NL	NL	NL	NL
	3.0.2	11/04	1.00	1.00	1.00	NL	1.05	1.00	NL	NL	NL	NL	NL
	3.1.0	06/05	1.02	1.00	1.00	1.01	1.05	1.00	NL	NL	NL	NL	NL

Model	Firmware		Lingoes										
	Version	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x0A	0x0C
4G (color display)	1.0.0	10/04	1.00	1.00	1.00	NL	1.05	1.00	NL	NL	NL	NL	NL
	1.1.0	03/05	1.01	1.00	1.00	1.00	1.06	1.00	NL	NL	NL	NL	NL
	1.2.0	06/05	1.02	1.00	1.00	1.01	1.06	1.00	NL	NL	NL	NL	NL
nano	1.0.0	09/05	1.02	NL	1.00	1.02	1.07	1.00	NL	1.00	1.00	NL	NL
	1.1.0	01/06	1.03	NL	1.00	1.03	1.09	1.00	NL	1.00	1.00	NL	NL
	1.1.1	03/06	1.03	NL	1.00	1.03	1.09	1.00	NL	1.00	1.00	NL	NL
	1.2.0	06/06	1.04	NL	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 ²	NL
	1.3	10/06	1.05	NL	1.02	1.05	1.11	1.01	NL	1.00	1.00	1.01	NL
5G	1.0.0	10/05	1.03	1.01	1.01	1.03	1.08	1.00	1.00	1.00	1.00	NL	NL
	1.1.0	01/06	1.03	1.01	1.01	1.03	1.09	1.00	1.00	1.00	1.00	NL	NL
	1.1.1	03/06	1.03	1.01	1.01	1.03	1.09	1.00	1.00	1.00	1.00	NL	NL
	1.1.2	06/06	1.03	1.01	1.01	1.03	1.09	1.00	1.00	1.00	1.00	NL	NL
	1.2.0	09/06	1.05	1.01	1.02	1.05	1.11	1.01	1.00	1.00	1.00	1.01	NL
	1.2.1	11/06	1.05	1.01	1.02	1.05	1.11	1.01	1.00	1.00	1.00	1.01	NL
	1.2.3	11/07	1.06	1.01	1.02	1.05	1.12	1.01	1.00	1.00	1.00	1.01	1.01
2G nano	1.0.0	09/06	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 ²	NL
	1.0.1	09/06	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 ²	NL
	1.0.2	09/06	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 ²	NL
	1.1	10/06	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 ²	NL
	1.1.1	10/06	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 ²	NL
	1.1.2	02/07	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.02	NL
classic	1.0	09/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.1	09/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.2	10/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.3	11/07	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.01
	1.1.1	02/08	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.01

Model	Firmware		Lingoes										
	Version	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x0A	0x0C
3G nano	1.1.2	05/08	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.02
	1.0	09/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.1	09/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.2	10/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.3	11/07	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.01
	1.1	01/08	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.01
	1.1.2	05/08	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.02

¹ Supporting the 3G iPod requires special design considerations. See ["The 3G iPod"](#) (page 595) and ["Interfacing With the 3G iPod"](#) in *MFi Accessory Hardware Specification*.

² Because version 1.00 of Lingo 0x0A contained a bug that was corrected in version 1.01, accessories should attempt Digital Audio only with Apple devices whose Lingo 0x0A version is higher than or equal to 1.01. See [Table 3-246](#) (page 350).

Table F-2 Past Apple device models, firmware, and lingo versions, table 2

Model	Firmware		Lingoes											
	Vers.	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x07	0x08	0x09	0x0A	0x0C	0x0E
iPhone	1.1	09/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.1	09/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.2	11/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.3	01/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.4	02/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.5	07/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.0	07/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.0.1	08/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.1.0	09/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	NL
	2.2.1	01/09	1.08	1.02	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	NL
	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
	3.1.2	10/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	1.00




Model	Firmware		Lingoes											
	Vers.	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x07	0x08	0x09	0x0A	0x0C	0x0E
1G touch	1.1	09/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.1	09/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.2	11/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.3	01/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.4	02/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.5	07/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.0	07/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.0.1	08/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.1.0	09/08	1.08	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	NL
	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
	3.1.2	10/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	1.00
iPhone 3G	2.1.0	09/08	1.08	1.02	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	NL
	2.2.1	01/09	1.08	1.02	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	NL
	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
	3.1.2	10/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	1.00
4G nano	1.0.2	09/08	1.08	1.02	1.03	1.05	1.14	1.01	1.01	1.00	1.01	1.03	1.02	NL
	1.0.3	01/09	1.08	1.02	1.04	1.05	1.14	1.01	1.01	1.00	1.01	1.03	1.02	NL
120 GB classic	2.0.0	09/08	1.08	1.02	1.02	1.05	1.13	1.01	1.00	1.00	NL	1.03	1.02	NL
	2.0.1	01/09	1.08	1.02	1.03	1.05	1.13	1.01	1.00	1.00	NL	1.03	1.02	NL
2G touch	2.1.1	09/08	1.08	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	NL
	2.2.1	01/09	1.08	1.02	1.02	1.05	1.12	1.01	NL	1.01	NL	1.02	1.02	NL
	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
iPhone 3GS	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
	3.1.2	10/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
160 GB classic	2.0.3	09/09	1.08	1.02	1.03	1.05	1.13	1.01	1.00	1.00	NL	1.03	1.02	NL

Model	Firmware		Lingoes											
	Vers.	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x07	0x08	0x09	0x0A	0x0C	0x0E
5G nano	1.0.1	09/09	1.08	1.02	1.04	1.05	1.14	1.01	1.01	1.00	1.01	1.03	1.02	NL

The 3G iPod

The *iPod Accessory Protocol Interface Specification* no longer covers the 3G iPod. The 3G iPod is the white iPod with 4 buttons above a white click wheel, shipped in April, 2003, shown in [Table F-11](#) (page 602). This section is included for historical purposes only.

Table F-3 3G iPod product identification

3G iPod		
	Product name: iPod (3rd generation)	Shipped: 04/2003
	Compatibility icons: <div>  <div> iPod 3rd generation 10GB 15GB 20GB </div> </div> <div>  <div> iPod 3rd generation 30GB 40GB </div> </div>	
	Connectivity: Dock connector, headphone jack	

General Compatibility With Accessories

3G iPod firmware version 2.0.0 supports only the small packet format with a maximum payload of 127 bytes; it does not support commands that require a larger payload. See ["Command Packet Formats"](#) (page 76).

3G iPod firmware versions 2.1.0 and later include limited support for the Accessory Power lingo (Lingo 0x05), but only through the 9-pin audio/remote connector. They do not support this lingo through the 30-pin connector.

Accessory Identification and iAP Commands

The 3G iPod does not support the IDPS process or the `IdentifyDeviceLingoes` command; it supports only the older `RequestIdentify` and `Identify` commands. See ["General Lingo Command 0x01: Identify \(Deprecated\)"](#) (page 598). In addition, the 3G iPod does not support authentication.

The 3G iPod does not return a response to an `Identify` command; an attached accessory must use another mechanism to determine whether the iPod has successfully processed the command.

The 3G iPod does not support the Display Remote lingo (Lingo 0x03).

The 3G iPod has limited functionality when executing certain database navigation commands in Extended Interface mode:

- `SelectDBRecord` does not support the track category.
- `RetrieveCategorizedDatabaseRecords` does not support a record count of –1.
- `ResetDBSelection` does not clear the database sort order.

User Interface Restrictions

The 3G iPod does not handle Fast Forward and Rewind notifications properly (see ["Command 0x09: RemoteEventNotification"](#) (page 237)).

The 3G iPod does not support podcasts or audiobooks.

9-Pin Audio/Remote Connector Commands

The commands documented in this section apply only to the 9-pin Audio/Remote connector. For the top connector microphone only, the Apple device sends a `BeginRecord` command when recording is about to begin. The accessory microphone bias, if applicable, should already be present. The Apple device sends an `EndRecord` command when recording is completed. The accessory may remove microphone bias, if applicable, after the `EndRecord` command is received.

The Apple device sends a `BeginPlayback` command when playback is about to begin. The accessory may then turn on its speaker amplifier, if present. Upon receipt of an `EndPlayback` command, the accessory must turn off its speaker amplifier. For all Microphone lingo commands sent by the Apple device, no accessory response is expected.

Command 0x00: BeginRecord

Direction: Apple device to Accessory

The Apple device sends this command to notify the accessory that audio recording has started. The accessory does not return a packet to the Apple device in response to this command. See ["Command 0x06: iPodModeChange"](#) (page 605) for more details.

Table F-4 `BeginRecord` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo

Byte number	Value	Comment
4	0x00	Command ID: BeginRecord
5	0xFD	Checksum

Command 0x01: EndRecord

Direction: Apple device to Accessory

The Apple device sends this command to notify the accessory that audio recording has ended. The accessory does not return a packet to the Apple device in response to this command. See "[Command 0x06: iPodModeChange](#)" (page 605) for more details.

Table F-5 EndRecord packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x01	Command ID: EndRecord
5	0xFC	Checksum

Command 0x02: BeginPlayback

Direction: Apple device to Accessory

The Apple device sends this command to notify the accessory that audio playback has started. The accessory does not return a packet to the Apple device in response to this command. See "[Command 0x06: iPodModeChange](#)" (page 605) for more details.

Table F-6 BeginPlayback packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x02	Command ID: BeginPlayback

Byte number	Value	Comment
5	0xFB	Checksum

Command 0x03: EndPlayback

Direction: Apple device to Accessory

The Apple device sends this command to notify the accessory that audio playback has ended. The accessory does not return a packet to the Apple device in response to this command. See ["Command 0x06: iPodModeChange"](#) (page 605) for more details.

Table F-7 EndPlayback packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x03	Command ID: EndPlayback
5	0xFA	Checksum

General Lingo Command 0x01: Identify (Deprecated)

Direction: Accessory to Apple device

This command is deprecated. It should be used only by accessories that need to support the 3G iPod. Supporting the 3G iPod requires other special design considerations; see "Interfacing With the 3G iPod" in *MFi Accessory Hardware Specification*.

Accessories that use this command send it to notify the Apple device that an accessory has been attached and to register the lingo it supports. Accessories should identify at boot time and any time they receive ["Command 0x00: RequestIdentify"](#) (page 83) from the Apple device. The Apple device does not send an ACK command in response.

Note: The `Identify` command should be used only over the UART serial port link.

Accessories should follow the identification guidelines in ["Packet Signaling and Initialization Using the UART Serial Port Link"](#) (page 66) and ["iAP Signaling and Initialization Using the USB or BT Port Link"](#) (page 69) to guarantee they have established communication with the Apple device when using this command. Accessory accessories that support more than one lingo (not including the General lingo) or that plan to use the USB transport should use the IDPS process.

The `Identify` command disables all but free lingo on the current port. For serial ports, this means lingo 0x00, 0x02, 0x03, and 0x05 may be used, excluding authenticated commands; the USB port will be able to use only the general lingo, 0x00 (see [Table 2-4](#) (page 72)). Devices that register with this command can use only the General lingo (0x00) commands plus those of the specific lingo that they identified, with a few exceptions. All accessories may use the Simple Remote lingo (0x02) `ContextButtonStatus` command (0x00). If the identified lingo is the Extended Interface lingo (0x04), the accessory may also use the Display Remote lingo (0x03) commands if that lingo is not already being used by another accessory.

If the lingo identified by the `Identify` command can be used by only one accessory at a time, and that lingo is already in use by a different accessory, the command will fail but no command failure `ACK` command will be sent to the accessory. The accessory can verify that the `Identify` command has succeeded by sending a free command with valid parameters and checking the Apple device's response. The Apple device will respond with an `ACK` response with failure status if the lingo is already in use.

This command performs lingo conflict checking to ensure that single-instance lingo, such as Display Remote, are used on only one port at a time.

Table F-8 Identify packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x01	Command: <code>Identify</code>
5	0x0N	Supported lingo. For example, if the accessory supports the Simple Remote lingo, this byte should be 0x02.
6	0xNN	Checksum

The `Identify` command has facilities for accessories to draw more than 5 mA power from the Apple device. The `Identify` command sent by such an accessory contains the supported lingo and the optional power bitfields described in [Table F-21](#) (page 608). These bits define the power requirements of the accessory. [Table F-20](#) (page 607) shows the format of an `Identify` command for a high-power accessory.

Table F-9 Identify packet for high-power accessories

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Packet payload length
3	0x00	Lingo ID: General lingo
4	0x01	Command: <code>Identify</code>

Byte number	Value	Comment
5	0x05	Supported lingo: Accessory Power lingo
6	0x00	Reserved: set to 0x00
7	0x02	Number of valid bits in the power option flag
8	0x0N	Option flag bits. See Table F-21 (page 608).
9	0xNN	Checksum

The option flag byte consists of bitfields. At this time, the most significant 6 bits of the option flags are reserved and should be zero. Bit 1 is defined for Apple use only and must be zero. Bit 0 should be 1 if the accessory requires more than 5 mA at any time. This may be the case if it is powered by the Apple device.

Table F-10 Power option bits

Bit	Description
0	Power consumption requirements. Possible values are: 0 = the accessory requires 5 mA or less power from Apple device at all times 1 = the accessory requires more than 5 mA power from Apple device (up to 100mA maximum) during playback operation
1	Reserved. This bit must be set to 0 by external accessories.
2–7	Reserved. Set to 0.

Deprecated Lingo 0x01: Microphone Lingo

Note: This lingo is deprecated; do not use it in new products.

The Microphone lingo enables combination microphone and speaker accessory devices to record and play back audio. Initial microphone devices supported one input mode (mono) and one sample rate (8 kHz). The increased Apple device mass storage disk capacities enable the option of supporting a stereo input mode and higher audio sample rates. With these changes, Apple devices may be used for high-quality mobile audio recording.

Note: The 5G iPod, 2G nano, iPod classic, and 3G nano support both stereo and monophonic microphones through the 30-pin connector. The 4G iPod only supports monophonic microphones plugged into the 9-pin audio/remote connector; it cannot record audio from a microphone connected to the 30-pin connector. See “The 9-Pin Audio/Remote Connector” in the chapter “Historical Information” in *MFi Accessory Hardware Specification*.

The Microphone lingo is defined such that the Apple device initiates commands and the accessory responds to these commands; that is, the Apple device sends commands to the accessory and the accessory responds with data or ACK commands.

When the Apple device detects an accessory speaking the Microphone lingo, it may transition into a recorder application where it can create and manage recordings. Based on the microphone accessory capabilities, the Apple device recording application may choose to change its appearance based on the presence or absence of certain microphone features. The accessory should indicate its capabilities to the Apple device on request. These capabilities may include:

- Stereo line input source
- Stereo/mono control
- Recording level control
- Recording level limiter

Note: Accessories using Authentication 2.0 and identifying for the Microphone lingo may have audio routed to them by default, based on their resistor ID, identification method and/or Apple device model. If a microphone accessory does not support line-out (i.e. it does not have a built-in speaker) and it uses IDPS for identification, its `AccCapsToken` must not confirm line-out support (see [Table 2-79](#) (page 133)).

Microphone accessory devices can draw power from the Apple device or supply power to the Apple device. Accessory power management is important as Apple devices transition to a smaller physical size at the same time as trying to extend battery life. An accessory using the Microphone lingo must remain in low power mode, as defined in the section “Accessory Power Policy” in *MFi Accessory Hardware Specification*. It is allowed to transition to high-power mode only if it receives an `iPodModeChange` command with a Mode value of 0x00 or 0x02 (begin audio recording or playback), as listed in [Table F-18](#) (page 606), and also only if it has declared the intermittent high-power option during its identification process. The accessory must return to low power mode within 100 ms of receiving an `iPodModeChange` command with a Mode value of 0x01 or 0x03 (end audio recording or playback). Like all accessories, an accessory using the Microphone lingo must comply with the power requirements of the Apple device’s Sleep and Hibernate states, as specified in Appendix A, “Apple Device Power States and Accessory Power,” in *MFi Accessory Hardware Specification*. Accessories are informed of Apple device state changes by receiving a General lingo `NotifyiPodStateChange` command.

The microphone accessory is responsible for keeping its power consumption below the maximum allowed limits for each Apple device state; see “Accessory Power Policy” in *MFi Accessory Hardware Specification*. Current from the Apple device on the Accessory Power line of the 30-pin connector is completely shut off when an Apple device enters the Hibernate state. On reset or power up, the accessory must be in low power mode with the amplifier off (that is, with audio input and output disabled).

Microphone state information must be retained locally by the accessory while uninterrupted current from the Apple device (either high or low power) is available. If current from the Apple device is turned off, accessory state information may be lost. Devices are not expected to retain state information across such power down cycles (Hibernate mode).

Level changes in Apple device playback volume may require the accessory to support Display Remote lingo (0x03) functionality.

Table F-11 (page 602) lists the commands available as part of the Microphone lingo.

Note: Legacy Microphone lingo commands 0x00–0x03 are disabled for devices using the 30-pin connector. They are superseded by "Command 0x06: iPodModeChange" (page 605), which returns an ACK. Deprecated commands 0x00–0x03 are documented in "9-Pin Audio/Remote Connector Commands" (page 596).

Table F-11 Microphone lingo command summary

Command	ID	Data length	Protocol Version	Connector	Authentication Required
BeginRecord	0x00	0x00	All	9-pin Audio/Remote	No
EndRecord	0x01	0x00	All	9-pin Audio/Remote	No
BeginPlayback	0x02	0x00	All	9-pin Audio/Remote	No
EndPlayback	0x03	0x00	All	9-pin Audio/Remote	No
ACK	0x04	0x02	1.01	30-pin	Yes
GetDevAck	0x05	0x00	1.01	30-pin	Yes
iPodModeChange	0x06	0x01	1.01	30-pin	Yes
GetDevCaps	0x07	0x00	1.01	30-pin	Yes
RetDevCaps	0x08	0x04	1.01	30-pin	Yes
GetDevCtrl	0x09	0x00	1.01	30-pin	Yes
RetDevCtrl	0x0A	0x02	1.01	30-pin	Yes
SetDevCtrl	0x0B	0x02	1.01	30-pin	Yes
Reserved	0x0C–0xFF	N/A	N/A	N/A	N/A

Table F-12 Select Microphone lingo command timings

Command	Timeout	Tries
GetDevAck (0x05)	200 ms	1
GetDevCaps (0x07)	200 ms	1
GetDevCtrl (0x09)	200 ms	1
SetDevCtrl (0x0B)	200 ms	1

Command History of the Microphone Lingo

Table F-13 (page 603) shows the history of command changes in the Microphone lingo.

Table F-13 Microphone lingo command history

Lingo version	Command changes	Features
No version	Add: 0x00–0x03	Microphone begin/end record/playback notification commands, 9-pin Audio/Remote connector only
1.00	None	Version number available through <code>RequestLingoProtocol - Version</code>
1.01	Add: 0x04–0x0B	ACK, mode change, capabilities, and control support (30-pin connector only)

The commands described in the following sections are used only with the 30-pin connector. For information about 9-pin connector commands, see ["9-Pin Audio/Remote Connector Commands"](#) (page 596).

Note: Devices must return responses to Apple device commands in the order in which the commands were received and within the specified time limits. Failing to send responses in the order commands were received or exceeding the command timeout limits could cause a communications failure and result in the accessory being considered not present by the Apple device.

Command 0x04: ACK

Direction: Accessory to Apple device

The microphone accessory sends this command in response to a command sent from the Apple device. Note that the commands 0x00–0x03 do not require an ACK response. The accessory sends an ACK response when a command that does not return any data has completed, a bad parameter is received, or an unsupported or invalid command is received.

Table F-14 ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x04	Command ID: ACK
5	0xNN	The command result status. See Table F-15 (page 604) for the possible values.
6	0xNN	The ID of the command for which the response is being sent.

Byte number	Value	Comment
7	0xNN	Checksum

Table F-15 (page 604) shows the possible values of the command result status byte.

Table F-15 Command result values

Byte	Meaning
0x00	Success (OK)
0x01	Not applicable: the Microphone lingo does not use this error value.
0x02	ERROR: Command failed. Sent in response to a valid command if that command did not succeed.
0x03	ERROR: Out of resources. This indicates that an internal allocation failed in the Apple device.
0x04	ERROR: Bad parameter. The command or input parameters are invalid.
0x05	Not applicable: the Microphone lingo does not use this error value.
0x06	Reserved
0x07	ERROR: The accessory is not authenticated to use this lingo command.
0x08	ERROR: Mismatched authentication protocol version.
0x09 - 0xFF	Reserved

Command 0x05: GetDevAck

Direction: Apple device to Accessory

The Apple device sends this command to get an ACK response from a microphone accessory. The Apple device uses this command to “ping” the accessory and determine that it is present and ready to accept commands. In response, the accessory sends the ACK command with command status OK.

The timeout for this command is 200 ms (0.2 second). The accessory must respond within the allotted time; the Apple device will not retry the command. If the accessory does not respond within the specified time, the command will fail and the accessory may be considered not present by the Apple device.

Table F-16 GetDevAck packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload

Byte number	Value	Comment
3	0x01	Lingo ID: Microphone lingo
4	0x05	Command ID: GetDevAck
5	0xF8	Checksum

Command 0x06: iPodModeChange

Direction: Apple device to Accessory

The Apple device sends this command to the microphone accessory when an audio recording or playback event occurs. The microphone accessory uses the `iPodModeChange` command to configure its inputs or outputs and power consumption level for the specified mode. In response, the accessory sends the `ACK` command with the command status OK. The accessory sends the `ACK` command when the accessory has completed its mode change.

The Apple device does not wait to receive an `ACK` command from the accessory in response to the mode change. It may continue sending other commands to the accessory after it has sent the mode change command.

Table F-17 `iPodModeChange` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x06	Command ID: <code>iPodModeChange</code>
5	0xNN	Mode. See Table F-18 (page 606).
6	0xNN	Checksum

[Table F-18](#) (page 606) lists the possible values of the Mode byte.

Table F-18 Mode values

Value	Meaning
0x00	<p>Begin audio recording mode.</p> <p>When it receives this command, the accessory can activate its microphone recording inputs or outputs connected to the Apple device's line inputs. If the accessory has requested the intermittent high power option using General lingo "Command 0x13: IdentifyDeviceLingoes" (page 96), it must wait until after this command is received before leaving low power mode, as defined in the section "Accessory Power Policy" in <i>MFi Accessory Hardware Specification</i>. By the time the accessory receives this command, accessory high power is enabled and ready to use during the recording process.</p>
0x01	<p>End audio recording mode.</p> <p>When it receives this command, the accessory can deactivate its microphone recording inputs or outputs and return to a quiescent state. If the accessory has requested the intermittent high power option, by using the General lingo command <code>IdentifyDeviceLingoes</code>, it may be in a high power consumption state. After receiving this command, the accessory must reduce its power consumption to low power mode, as defined in the section "Accessory Power Policy" in <i>MFi Accessory Hardware Specification</i>, within 100 ms.</p>
0x02	<p>Begin audio playback mode.</p> <p>When it receives this command, the accessory can activate its speaker playback inputs or outputs connected to the Apple device's line outputs, if present. If the accessory has requested the intermittent high power option, by using the General lingo command <code>IdentifyDeviceLingoes</code>, it must wait until after this command is received before consuming more than low power. By the time the accessory receives this command, accessory high power is enabled and ready to use during the playback process.</p>
0x03	<p>End audio playback mode.</p> <p>When it receives this command, the accessory can deactivate its speaker playback inputs or outputs and return to a quiescent state. If the accessory has requested the intermittent high power option using the General lingo command <code>IdentifyDeviceLingoes</code>, it may be in a high power consumption state. After receiving this command, the accessory must reduce its power consumption below low power mode, as defined in the section "Accessory Power Policy" in <i>MFi Accessory Hardware Specification</i>, within 100 ms.</p>
0x04–0xFF	Reserved.

Note: Failure to wait for a mode change notification before increasing power consumption could result in an incompatibility with present and future Apple devices.

Failure to reduce power consumption within the stated time could result in an incompatibility with present and future Apple devices.

Command 0x07: GetDevCaps

Direction: Apple device to Accessory

The Apple device sends this command to the microphone accessory to determine the features present on the accessory. In response, the accessory sends "[Command 0x08: RetDevCaps](#)" (page 607) with the payload indicating the capabilities it supports.

The timeout for this command is 200 ms (0.2 second). The accessory must respond within the allotted time; the Apple device will not retry the command. If the accessory does not respond within the specified time, the command will fail and the accessory may be considered not present by the Apple device.

Table F-19 GetDevCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x07	Command ID: GetDevCaps
5	0xF6	Checksum

Command 0x08: RetDevCaps

Direction: Accessory to Apple device

The accessory sends this command in response to the command "[Command 0x07: GetDevCaps](#)" (page 606) sent by the Apple device. The microphone accessory returns the payload indicating which capabilities it supports.

Table F-20 RetDevCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x08	Command ID: RetDevCaps
5	0xNN	Accessory capabilities (bits 31: 24). See Table F-21 (page 608).
6	0xNN	Accessory capabilities (bits 23: 16)
7	0xNN	Accessory capabilities (bits 15: 8)
8	0xNN	Accessory capabilities (bits 7: 0)

Byte number	Value	Comment
9	0xNN	Checksum

The capabilities bit ranges correspond to the microphone control commands. The Apple device should not attempt to control accessory features, using "Command 0x0B: SetDevCtrl" (page 613), if the associated capabilities bits are not set. Table F-21 (page 608) lists the meaning of these bits.

Table F-21 Microphone capabilities bitmask

Bit	Meaning
00	Stereo line input. A value of 0 indicates the accessory is monophonic only.
01	Stereo or mono line input. This bit should be set only if the microphone supports stereo line input and can switch between stereo and mono modes.
02	Recording level is present and variable.
03	Recording level limit is present.
04	Accessory supports duplex audio; it can play audio output from the Apple device while it sends audio input to the Apple device.
31:05	Reserved.

Command 0x09: GetDevCtrl

Direction: Apple device to Accessory

The Apple device sends this command to get the accessory control state for the specified control type. In response, the accessory sends "Command 0x0A: RetDevCtrl" (page 609) with its current control state. If this command is not supported by the accessory—that is, if the microphone does not have any configurable controls—it must return an ACK command with a bad parameter error status.

The timeout for this command is 200 ms (0.2 second). The accessory must respond within the allotted time; the Apple device will not retry the command. If the accessory does not respond within the specified time, the command will fail and the accessory may be considered not present by the Apple device.

Table F-22 GetDevCtrl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x09	Command ID: GetDevCtrl

Byte number	Value	Comment
5	0xNN	The control type for which to get the state. The possible values are: 0x00: Reserved. 0x01: Stereo/mono line input. 0x02: Recording level control. 0x03: Recording level limiter control. 0x04–0xFF: Reserved.
6	0xNN	Checksum

Command 0x0A: RetDevCtrl

Direction: Accessory to Apple device

The accessory sends this command in response to the command "[Command 0x09: GetDevCtrl](#)" (page 611) received from the Apple device. The accessory returns the current control state for the specified control type. Control types are supported only if the associated capabilities bits are set in the command "[Command 0x08: RetDevCaps](#)" (page 607).

Table F-23 RetDevCtrl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x0A	Command ID: RetDevCtrl
5	0xNN	The control type. See Table F-24 (page 609).
6	0xNN	The control data. See Table F-24 (page 609).
7	0xNN	Checksum

[Table F-24](#) (page 609) lists the different control types and the data associated with them.

Table F-24 Control types and data

Control type value	Control type	Control data
0x00	Reserved	Stereo capability cannot be set.

Control type value	Control type	Control data
0x01	Stereo/mono line input	Possible data values are: 0x00 = mono 0x01 = stereo 0x02–0xFF = reserved
0x02	Recording level control	Possible data values are in a range between: 0x00 = mute 0xFF = maximum gain
0x03	Recording level limiter	Possible data values are: 0x00 = off 0x01 = on 0x02–0xFF = reserved
0x04–0xFF	Reserved	

Command 0x0B: SetDevCtrl

Direction: Apple device to Accessory

The Apple device sends this command to set the accessory control state for the specified control type. In response, the accessory sends the `ACK` command with the command status. If this command is not supported by the accessory—that is, if the microphone does not have any configurable controls—it must return an `ACK` command with a bad parameter error status.

The timeout for this command is 200 ms (0.2 second). The accessory must respond within the allotted time; the Apple device will not retry the command. If the accessory does not respond within the specified time, the command will fail and the accessory may be considered not present by the Apple device.

Table F-25 SetDevCtrl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x0B	Command ID: SetDevCtrl
5	0xNN	The control type. The control type and data values are the same as for the RetDevCtrl (0x0A) command. See Table F-24 (page 609) for more information.

Byte number	Value	Comment
6	0xNN	The control data. See Table F-24 (page 609) for more information.
7	0xNN	Checksum

Deprecated MicrophoneCapsToken

Table F-26 MicrophoneCapsToken format

Name	Bytes	Value	Description
length	1	0x06	Length of this token-value field in bytes, not including this byte.
FIDType	1	0x01	First byte of token ID.
FIDSubtype	1	0x00	Second byte of token ID.
micCapsBitmask	4	0xNNNNNNNN	Microphone capabilities; see Table F-27 (page 611).

Table F-27 Microphone capabilities bits

Bit	Meaning
00	Stereo line input. A value of 0 indicates that the accessory is monophonic only.
01	Stereo or mono line input. This bit should be set only if the microphone supports stereo line input and can switch between stereo and mono modes.
02	Recording level is present and variable.
03	Recording level limit is present.
04	Accessory supports duplex audio; it can play audio output from the Apple device while it sends audio input to the Apple device.
31:05	Reserved.

Deprecated Simple Remote Lingo Command

Note: This command is deprecated; do not use it in new products.

Command 0x02: ImageButtonStatus

Direction: Accessory to Apple device

The accessory sends this command to the Apple device when an image-specific button event occurs. The button status is a bitmask representing each button that is currently pressed. The button status packet must be repeatedly sent by the accessory at intervals between 30 and 100 ms while one or more buttons are pressed. When all buttons are released, the accessory must send a button status packet with a 0x00 payload to indicate that no buttons are pressed. In response, the Apple device will return an ACK packet containing the command status to the accessory.

Note: This command is not supported in Apple products running iOS 3.2.

Table F-28 ImageButtonStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x02	Command ID: ImageButtonStatus
5	0xNN	Byte index 0, image-specific button states 7:0. See Table F-29 (page 612) for a list of image-specific button states recognized by the Apple device.
6	0xNN	Byte index 1, button states 15:8 (optional)
7	0xNN	Byte index 2, button states 23:16 (optional)
8	0xNN	Byte index 3, button states 31:24 (optional)
(last byte)	0xNN	Checksum

Table F-29 Image-specific button values

Button name	Number	Byte index	Button bitmask
Play/Pause	0	0x00	0x01
Next image	1	0x00	0x02
Previous image	2	0x00	0x04
Stop	3	0x00	0x08
Play/resume	4	0x00	0x10
Pause	5	0x00	0x20
Shuffle advance	6	0x00	0x40
Repeat advance	7	0x00	0x80

Button name	Number	Byte index	Button bitmask
Reserved	15:8	0x01	0xFF
Reserved	23:16	0x02	0xFF
Reserved	31:24	0x03	0xFF

Deprecated USB Host Control Commands

Note: These commands are deprecated; do not use them in new products. For USB Host Mode functionality, see “[Lingo 0x06: USB Host Mode Lingo](#)” (page 273).

Command 0x00: ACK

Direction: Accessory to Apple device

This command is sent by the accessory in response to a command received from the Apple device. It reports the original command number and the status of the command.

Table F-30 ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x06	Lingo ID: USB Host Control lingo
4	0x00	Command ID: ACK
5	0xNN	Command status (see Table 3-17 (page 201))
6	0xNN	ID of the command being acknowledged
7	0xNN	Checksum

Command 0x01: GetUSBPowerState

Direction: Apple device to Accessory

This command is sent by the Apple device to obtain the accessory's current USB power state. In response, the accessory must send a `RetUSBPowerState` command with the current USB power setting.

Table F-31 GetUSBPowerState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x06	Lingo ID: USB Host Control lingo
4	0x01	Command ID: GetUSBPowerState
5	0xF7	Checksum

Command 0x02: RetUSBPowerState

Direction: Accessory to Apple device

This command is sent by the accessory in response to a GetUSBPowerState command received from the Apple device. It returns the current state of the USB power supply.

Table F-32 RetUSBPowerState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x06	Lingo ID: USB Host Control lingo
4	0x02	Command ID: RetUSBPowerState
5	0xNN	Current power state (zero for off, nonzero for on)
6	0xNN	Checksum

Command 0x03: SetUSBPowerState

Direction: Apple device to Accessory

This command is sent by the Apple device to set the accessory's USB power state. The accessory must set the USB power state and respond with an ACK command indicating command completion.

Table F-33 SetUSBPowerState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x06	Lingo ID: USB Host Control lingo
4	0x03	Command ID: SetUSBPowerState
5	0xNN	Power state to be set (zero for off, nonzero for on)
6	0xNN	Checksum

Deprecated Extended Interface Commands

The commands in this section have been replaced by commands in the iAP General lingo (Lingo 0x00).

Command 0x0012: RequestProtocolVersion

Direction: Accessory to Apple device

Note: This command is deprecated except for use by accessories supporting 3G iPods. Use the General lingo (0x00) command 0x0F, RequestLingoProtocolVersion, instead.

Requests the version of the running protocol from the Apple device. The Apple device responds with a "Command 0x0013: ReturnProtocolVersion" (page 616) command.

Note: This command requests the Extended Interface protocol version, not the Apple device software version.

Table F-34 RequestProtocolVersion command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)

Byte number	Value	Meaning
5	0x12	Command ID (bits 7:0)
6	0xE7	Packet payload checksum byte

Command 0x0013: ReturnProtocolVersion

Direction: Apple device to Accessory

Note: This command is deprecated except for use by accessories supporting 3G iPods. Use the General lingo (0x00) command 0x10, `ReturnLingoProtocolVersion`, instead.

Returns the Apple device Extended Interface protocol version number. The Apple device sends this command in response to the "[Command 0x0012: RequestProtocolVersion](#)" (page 615) command from the accessory. The major version number specifies the protocol version digits to the left of the decimal point; the minor version number specifies the digits to the right of the decimal point. For example, a major version number of 0x01 and a minor version number of 0x08 represents an Extended Interface protocol version of 1.08. This protocol information is also available through the General lingo (lingo 0x00) command `RequestLingoProtocolVersion` when passing lingo 0x04 as the lingo parameter.

Note: This command returns the Extended Interface protocol version, not the Apple device software version.

Table F-35 `ReturnProtocolVersion` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x05	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x13	Command ID (bits 7:0)
6	0xNN	Protocol major version number
7	0xNN	Protocol minor version number
8	0xNN	Packet payload checksum byte

Command 0x0014: RequestiPodName

Direction: Accessory to Apple device

Note: This command is deprecated except for use by accessories supporting 3G iPods. Use the General lingo (0x00) command 0x07, `RequestiPodName`, instead.

Returns the name of the user's Apple device or "iPod" if the Apple device name is undefined. This allows the Apple device name to be shown in the human-machine interface (HMI) of the interfacing body. The Apple device responds with the "[Command 0x0015: ReturniPodName](#)" (page 617) command containing the Apple device name text string.

Table F-36 `RequestiPodName` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x14	Command ID (bits 7:0)
6	0xE5	Packet payload checksum byte

Command 0x0015: ReturniPodName

Direction: Apple device to Accessory

Note: This command is deprecated except for use by accessories supporting 3G iPods. Use the General lingo (0x00) command 0x08, `ReturniPodName`, instead.

The Apple device sends this command in response to the "[Command 0x0014: RequestiPodName](#)" (page 616) command from the accessory. The Apple device name is encoded as a null-terminated UTF-8 character array. The Apple device name string is not limited to 252 characters; it may be sent in small or large packet format. The small packet format is shown.

Note: Starting with version 1.07 of the Extended Interface lingo, the `ReturniPodName` command on Windows-formatted Apple devices returns the iTunes name of the Apple device instead of the Windows volume name.

Table F-37 `ReturniPodName` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x15	Command ID (bits 7:0)
6...N	0xNN	Apple device name as UTF-8 character array
(last byte)	0xNN	Packet payload checksum byte

Command 0x0038: SelectSortDBRecord

Direction: Accessory to Apple device

Applies To: Database Engine

Note: This command is **deprecated**. Use `SelectDBRecord` instead.

This command acts the same as "[Command 0x0017: SelectDBRecord](#)" (page 458), but includes a sorting feature. It selects one or more records in the Apple device database, based on a category-relative index. For example, selecting category 2 (Artist), record index 1, and sort order 3 (Album) results in a list of selected tracks (records) from the second artist in the artist list, sorted by album name. Selections are additive and limited by the category hierarchy; see "[Database Category Hierarchies](#)" (page 419). Subsequent selections are made based on the subset of records resulting from previous selections and not from the entire database.

Table F-38 SelectSortDBRecord command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x09	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x38	Command ID (bits 7:0)
6	0xNN	Database category type. See Table 4-37 (page 460).
7	0xNN	Category record index (bits 31:24)
8	0xNN	Category record index (bits 23:16)
9	0xNN	Category record index (bits 15:8)

Byte number	Value	Meaning
10	0xNN	Category record index (bits 7:0)
11	0xNN	Database sort type. See Table F-39 (page 619).
12	0xNN	Packet payload checksum byte

[Table F-39](#) (page 619) shows the possible sort orders.

Table F-39 Database sort order options

Sort Order	Code	Protocol version
Sort by genre	0x00	1.00
Sort by artist	0x01	1.00
Sort by composer	0x02	1.00
Sort by album	0x03	1.00
Sort by name	0x04	1.00
Reserved	0x05	N/A
Sort by release date	0x06	1.08
Sort by series (video only)	0x07	1.12
Sort by season (video only)	0x08	1.12
Sort by episode (video only)	0x09	1.12
Reserved	0x0A – 0xFE	N/A
Use default sort type	0xFF	1.00

Authentication 1.0 Sample Command Sequence

For legacy purposes, [Table F-40](#) (page 620) shows the process implemented by some existing accessories, using authentication 1.0. This process is not supported for new accessory designs.

Table F-40 Legacy accessory authentication

Step	Action or command	Direction	Comments
1	IdentifyDeviceLingoes (0x13)	Acc to Dev	The accessory identifies itself, lists its supported lingoes, and requests authentication. It can request immediate authentication or it can defer authentication until it tries to use restricted lingoes or commands.
2	ACK (0x02)	Dev to Acc	The Apple device acknowledges receipt of IdentifyDeviceLingoes.
5	GetDevAuthentication-Info (0x14)	Dev to Acc	The Apple device requests accessory authentication information and starts its timeout timer.
6	RetDevAuthentication-Info (0x15)	Acc to Dev	The accessory returns its major and minor authentication version.
7	AckDevAuthentication-Info (0x16)	Dev to Acc	The Apple device returns the status of its check of the authentication version. If it does not support the authentication version, the Apple device sends a value of 0x08 to the accessory.
The Apple device lingoes and commands are enabled after the authentication version is validated.			
8	GetAccessoryInfo (0x27)	Dev to Acc	The Apple device queries the accessory for information about it.
9	RetAccessoryInfo (0x28)	Acc to Dev	The accessory returns information about its identity and capabilities.
10	GetDevAuthentication-Signature (0x17)	Dev to Acc	The Apple device sends a 16-byte random challenge to the accessory and asks it to calculate a digital signature.
11	RetDevAuthentication-Signature (0x18)	Acc to Dev	The accessory returns its digital signature to the Apple device within 7.5 seconds.
12	AckDevAuthentication-Status (0x19)	Dev to Acc	The Apple device verifies the signature by deriving a public key from the Device ID and returns the status of signature verification.

Accessory Identification With Non-IDPS Apple Devices

Some models of Apple devices do not support IDPS. Every accessory must always start its identification process by sending `StartIDPS`; if the Apple device's ACK response passes a status value of 0x04 (Bad Parameter), the accessory must revert to an identification process using `IdentifyDeviceLingoes`.

Note: The accessory must send `IdentifyDeviceLingoes` within 800 ms of receiving the Apple device's ACK response.

The sample command listings in this section illustrate various forms of the identification process using `IdentifyDeviceLingoes`:

- [Table F-41](#) (page 621) shows the basic process for accessories that communicate using the UART port link.
- [Table F-42](#) (page 622) shows the basic process for accessories that communicate using USB or Bluetooth.
- [Table F-43](#) (page 623) lists sample commands for an accessory that supports the General and Extended Interface lingoes. For Extended Interface command details, see ["The Extended Interface Protocol"](#) (page 409).
- [Table F-44](#) (page 628) lists sample commands for an accessory that supports the General, Simple Remote, and Display Remote lingoes.
- [Table F-45](#) (page 631) lists sample commands for an accessory that supports the General, Simple Remote, Display Remote, and Digital Audio lingoes.
- [Table F-46](#) (page 633) shows a sample sequence of commands that implements radio tagging in an accessory. See ["iTunes Tagging"](#) (page 535).
- [Table F-47](#) (page 636) shows a sample sequence of commands that implements the Nike + iPod cardio equipment system in an accessory. See ["Nike + iPod Cardio Equipment System"](#) (page 569).

Table F-41 UART accessory identification with non-IDPS Apple devices

Step	Action or command	Direction	Comments
1	Wait 80 ms after the Apple device turns on Accessory Power (pin 13 in "Hardware Interfaces" in <i>MFi Accessory Hardware Specification</i>)	Acc	Wait for the Apple device's internal bootstrap and wakeup. If the Apple device is in Sleep mode, the accessory must repeat Steps 1-5 until the Apple device wakes and the accessory receives an ACK command.
2	Send sync byte (0xFF)	Acc to Dev	Allow the Apple device to synchronize to the accessory's baud rate.
3	Wait 20 ms	Acc	
4	<code>StartIDPS</code> (0x38)	Acc to Dev	The accessory sends <code>StartIDPS</code> to start the identification process specified in "Accessory Identification" (page 519).
5	Wait up to 1 second	Acc	The accessory waits for the Apple device to send an ACK command acknowledging receipt of <code>StartIDPS</code> .
6	ACK (0x02) of <code>StartIDPS</code>	Dev to Acc	The Apple device sends a status of 0x04 (Bad Parameter) to indicate that it does not support IDPS; see Table 2-1 (page 67), Step 6.

Step	Action or command	Direction	Comments
7	IdentifyDeviceLingoes (0x13)	Acc to Dev	The accessory sends IdentifyDeviceLingoes with the lingo mask set to only the General lingo, the option bit mask set to 0x0, and the Device ID set to 0x0. This cancels any active authentication process, as detailed in "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99).
8	Wait up to 1 second	Acc	The accessory waits for the Apple device to send an ACK command acknowledging receipt of IdentifyDeviceLingoes.
9	ACK (0x02) of IdentifyDeviceLingoes	Dev to Acc	The Apple device sends a status of 0x00 (OK) to acknowledge receipt of the empty IdentifyDeviceLingoes command.
If the Apple device does not send an ACK command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step 7 up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an ACK command after the third try, it may either quit or send an Identify command to achieve limited use of iAP. See The 3G iPod (page 595) and "Interfacing With the 3G iPod" in <i>MFi Accessory Hardware Specification</i> .			
10	GetAccessoryInfo (0x27)	Dev to Acc	The Apple device queries the accessory for information about it. The accessory ignores this query.
11	IdentifyDeviceLingoes (0x13)	Acc to Dev	The accessory sends a second IdentifyDeviceLingoes command, specifying which lingoes it supports and requesting immediate authentication (see "Authentication" (page 71)).
12	ACK (0x02) of IdentifyDeviceLingoes	Dev to Acc	The Apple device acknowledges receipt of the second IdentifyDeviceLingoes command.
The Apple device now initiates the authentication process by sending a GetDevAuthenticationInfo command to the accessory, as shown in Table 2-5 (page 74).			

Table F-42 USB or BT accessory identification with non-IDPS Apple devices

Step	Action or command	Direction	Comments
1	StartIDPS (0x38)	Acc to Dev	The accessory sends StartIDPS to start the identification process specified in "Accessory Identification" (page 519).
2	Wait up to 1 second	Acc	The accessory waits for the Apple device to send an ACK command acknowledging receipt of StartIDPS.

Step	Action or command	Direction	Comments
3	ACK (0x02) of StartIDPS	Dev to Acc	The Apple device sends a status of 0x04 (Bad Parameter) to indicate that it does not support IDPS; see Table 2-2 (page 69), Step 3.
4	IdentifyDeviceLingoes (0x13)	Acc to Dev	The accessory sends IdentifyDeviceLingoes with the lingo mask set to only the General lingo, the option bit mask set to 0x0, and the Device ID set to 0x0. This cancels any active authentication process, as detailed in "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99).
5	Wait up to 1 second	Acc	The accessory waits for the Apple device to send an ACK command acknowledging receipt of IdentifyDeviceLingoes.
6	ACK (0x02) of IdentifyDeviceLingoes	Dev to Acc	The Apple device sends a status of 0x00 (OK) to acknowledge receipt of the empty IdentifyDeviceLingoes command.
7	GetAccessoryInfo (0x27)	Dev to Acc	The Apple device queries the accessory for information about it. The accessory ignores this query.
8	IdentifyDeviceLingoes (0x13)	Acc to Dev	The accessory sends a second IdentifyDeviceLingoes command, specifying which lingoes it supports and requesting immediate authentication (see "Authentication" (page 71)).
9	ACK (0x02) of IdentifyDeviceLingoes	Dev to Acc	The Apple device acknowledges receipt of the second IdentifyDeviceLingoes command.
The Apple device now initiates the authentication process by sending a GetDevAuthenticationInfo command to the accessory, as shown in Table 2-5 (page 74).			

Table F-43 Non-IDPS identification of lingoes 0x00+0x04

Step	Accessory command	Apple device command	Comment
Steps A–C cancel any current authentication process; see "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). Step B also detects if the attached Apple device is a 3G iPod.			
A	IdentifyDeviceLingoes		identifying lingo 'General'; options none; device ID 0x00000000
The accessory waits up to 1 second for the Apple device to respond.			
B		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'

Step	Accessory command	Apple device command	Comment
If the Apple device does not send an <code>ACK</code> command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an <code>ACK</code> command after the third try, it may either quit or send an <code>Identify</code> command to achieve limited use of iAP. See The 3G iPod (page 595) and “Interfacing With the 3G iPod” in <i>MFi Accessory Hardware Specification</i> .			
C		<code>GetAccessoryInfo</code>	querying the accessory; accessory info type = 0x00 (get accessory info capabilities). The accessory ignores this query.
The accessory is now assured that it can successfully identify its lingo to the attached Apple device.			
1	<code>IdentifyDevice-Lingoes</code>		identifying lingo 'General Extended Interface'; options 'auth:immediate; power:low; device ID 0x00000200
2		<code>ACK</code>	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
3		<code>GetDevAuthentication-Info</code>	no params
4	<code>RetDevAuthentication-Info</code>		returning auth protocol v2.0; section: 0/1;
5		<code>ACK</code>	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
6	<code>RetDevAuthentication-Info</code>		returning auth protocol v2.0; section: 1/1;
7		<code>AckDevAuthentication-Info</code>	acknowledging 'auth info supported'
8		<code>GetAccessoryInfo</code>	requesting 'Acc info capabilities'
9		<code>GetDevAuthentication-Signature</code>	offering challenge
10	<code>RetAccessoryInfo</code>		returning 'Acc info capabilities Acc name Acc FW version Acc HW version Acc manufacturer Acc model number Acc serial number Acc incoming max packet size' in response to 'Acc info capabilities'
11		<code>GetAccessoryInfo</code>	requesting 'Acc name'
12	<code>RetDevAuthentication-Signature</code>		returning signature

Step	Accessory command	Apple device command	Comment
13	RetAccessoryInfo		returning 'Apple' in response to 'Acc name'
14		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
15		GetAccessoryInfo	requesting 'Acc FW version'
16	RetAccessoryInfo		returning 'v9.8.7' in response to 'Acc FW version'
17		GetAccessoryInfo	requesting 'Acc HW version'
18	RetAccessoryInfo		returning 'v1.2.3' in response to 'Acc HW version'
19		GetAccessoryInfo	requesting 'Acc manufacturer'
20	RetAccessoryInfo		returning 'Apple Inc.' in response to 'Acc manufacturer'
21		GetAccessoryInfo	requesting 'Acc model number'
22	RetAccessoryInfo		returning 'ModelNumber-XYZ' in response to 'Acc model number'
23		GetAccessoryInfo	requesting 'Acc serial number'
24	RetAccessoryInfo		returning 'SerialNumber-1234' in response to 'Acc serial number'
25		GetAccessoryInfo	requesting 'Acc incoming max packet size'
26	RetAccessoryInfo		returning '1024 bytes' in response to 'Acc incoming max packet size'
27	EnterRemoteUIMode		no params
28		ACK	acknowledging 'Command Pending (max wait time: 3000 ms)' to command 'General Lingo::EnterRemoteUIMode'
29		ACK	acknowledging 'Success (OK)' to command 'General Lingo::EnterRemoteUIMode'
30	RequestRemoteUIMode		no params
31		ReturnRemoteUIMode	returning 'Extended Interface Mode'
32	ResetDBSelection		no params

Step	Accessory command	Apple device command	Comment
33		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::ResetDBSelection'
34	GetNumberCategorized-DBRecords		for category 'Playlist'
35		ReturnNumber-CategorizedDBRecords	returning 'record count 22'
36	RetrieveCategorized-DatabaseRecords		requesting 'category Playlist record start index 0 record read count 4'
37		ReturnCategorized-DatabaseRecord	returning 'record category index 0 MyiPhone
38		ReturnCategorized-DatabaseRecord	returning 'record category index 1 Purchased
39		ReturnCategorized-DatabaseRecord	returning 'record category index 2 Against Doctor's Orders
40		ReturnCategorized-DatabaseRecord	returning 'record category index 3 Always In Your Mind
41	SelectDBRecord		selecting 'type Playlist record index 3
42		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::SelectDBRecord'
43	PlayCurrentSelection		playing selection track index 0
44		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::PlayCurrentSelection'
45	SetPlayStatusChange-Notification		setting 'basic play state changes track index'
46		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::SetPlayStatusChangeNotification'
47	PlayControl		sending 'Toggle Play/Pause'
48		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::PlayControl'
49	PlayControl		sending 'Next Track'

Step	Accessory command	Apple device command	Comment
50		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::PlayControl'
51		PlayStatusChange-Notification	notifying new play status 'playback track changed' (new track record index 1)
52	PlayControl		sending 'Toggle Play/Pause'
53		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::PlayControl'
54	SetCurrentPlaying-Track		setting currently playing track to 3
55		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::SetCurrentPlayingTrack'
56		PlayStatusChange-Notification	notifying new play status 'playback track changed' (new track record index 3)
57	GetPlayStatus		no params
58		ReturnPlayStatus	returning 'Playing track length 155506 ms track position 15637 ms'
59	GetIndexedPlaying-TrackTitle		requesting title name for track index 3
60		ReturnIndexedPlaying-TrackTitle	returning Auld Lang Syne
61	GetIndexedPlaying-TrackArtistName		requesting artist name for track index 3
62		ReturnIndexedPlaying-TrackArtistName	returning Straight No Chaser
63	GetIndexedPlaying-TrackAlbumName		requesting album name for track index 3
64		ReturnIndexedPlaying-TrackAlbumName	returning Holiday Spirits (Bonus Track Version)
65	ExitRemoteUIMode		no params
66		ACK	acknowledging 'Command Pending (max wait time: 3000 ms)' to command 'General Lingo::ExitRemoteUIMode'

Step	Accessory command	Apple device command	Comment
67		ACK	acknowledging 'Success (OK)' to command 'General Lingo::ExitRemoteUIMode'

Table F-44 Non-IDPS identification of lingo 0x00+0x02+0x03

Step	Accessory command	Apple device command	Comment
Steps A–C cancel any current authentication process; see "Cancelling a Current Authentication Process With IdentifyDeviceLingo" (page 99). Step B also detects if the attached Apple device is a 3G iPod.			
A	IdentifyDevice-Lingo		identifying lingo 'General'; options none; device ID 0x00000000
The accessory waits up to 1 second for the Apple device to respond.			
B		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingo'
If the Apple device does not send an ACK command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an ACK command after the third try, it may either quit or send an <code>Identify</code> command to achieve limited use of iAP. See The 3G iPod (page 595) and "Interfacing With the 3G iPod" in <i>MFi Accessory Hardware Specification</i> .			
C		GetAccessoryInfo	the accessory ignores this query
The accessory is now assured that it can successfully identify its lingo to the attached Apple device.			
1	IdentifyDevice-Lingo		identifying lingo 'General Simple Remote Display Remote'; options 'auth:immediate; power:low; device ID 0x00000200
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingo'
3		GetDevAuthentication-Info	no params
4	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1;
5		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
6	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1;

Step	Accessory command	Apple device command	Comment
7		AckDevAuthentication-Info	acknowledging 'auth info supported'
8		GetAccessoryInfo	requesting 'Acc info capabilities'
9		GetDevAuthentication-Signature	offering challenge
10	RetAccessoryInfo		returning 'Acc info capabilities Acc name Acc FW version Acc HW version Acc manufacturer Acc model number Acc serial number Acc incoming max packet size' in response to 'Acc info capabilities'
11		GetAccessoryInfo	requesting 'Acc name'
12	RetDevAuthentication-Signature		returning signature
13	RetAccessoryInfo		returning 'Apple' in response to 'Acc name'
14		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
15		GetAccessoryInfo	requesting 'Acc FW version'
16	RetAccessoryInfo		returning 'v9.8.7' in response to 'Acc FW version'
17		GetAccessoryInfo	requesting 'Acc HW version'
18	RetAccessoryInfo		returning 'v1.2.3' in response to 'Acc HW version'
19		GetAccessoryInfo	requesting 'Acc manufacturer'
20	RetAccessoryInfo		returning 'Apple Inc.' in response to 'Acc manufacturer'
21		GetAccessoryInfo	requesting 'Acc model number'
22	RetAccessoryInfo		returning 'ModelNumber-XYZ' in response to 'Acc model number'
23		GetAccessoryInfo	requesting 'Acc serial number'
24	RetAccessoryInfo		returning 'SerialNumber-1234' in response to 'Acc serial number'
25		GetAccessoryInfo	requesting 'Acc incoming max packet size'

Step	Accessory command	Apple device command	Comment
26	RetAccessoryInfo		returning '1024 bytes' in response to 'Acc incoming max packet size'
27	SetRemoteEvent-Notification		setting 'Track playback index Play status'
28		ACK	acknowledging 'Success (OK)' to command 'Display Remote Lingo::SetRemoteEventNotification'
29	GetIndexedPlaying-TrackInfo		getting info type 'Track title' for track index 0 and chapter index 0
30		RetIndexedPlaying-TrackInfo	returning 'Always In Your Mind' for info type 'Track title'
31	GetPlayStatus		no params
32		RetPlayStatus	returning 'Playback paused index 3 length 155506 ms position 84933 ms'
33	ContextButtonStatus		sending status 'Play/Pause'
34	ContextButtonStatus		sending status 'All buttons up'
35		RemoteEvent-Notification	reporting status 'Playing' for parameter 'Play status'
36	GetPlayStatus		no params
37		RetPlayStatus	returning 'Playing index 3 length 155506 ms position 91644 ms'
38	ContextButtonStatus		sending status 'Next Track'
39	ContextButtonStatus		sending status 'All buttons up'
40		RemoteEvent-Notification	reporting status 'index 4' for parameter 'Track playback index'
41	SetCurrentPlaying-Track		setting index 0
42		ACK	acknowledging 'Success (OK)' to command 'Display Remote Lingo::SetCurrentPlayingTrack'
43		RemoteEvent-Notification	reporting status 'index 0' for parameter 'Track playback index'
44	GetIndexedPlaying-TrackInfo		getting info type 'Track title' for track index 0 and chapter index 0

Step	Accessory command	Apple device command	Comment
45		RetIndexedPlaying-TrackInfo	returning 'Always In Your Mind' for info type 'Track title'

Table F-45 Non-IDPS identification of lingo 0x00+0x02+0x03+0x0A

Step	Accessory command	Apple device command	Comment
Steps A–C cancel any current authentication process; see "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). Step B also detects if the attached Apple device is a 3G iPod.			
A	IdentifyDevice-Lingoes		identifying lingo 'General'; options none; device ID 0x00000000
The accessory waits up to 1 second for the Apple device to respond.			
B		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
If the Apple device does not send an ACK command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an ACK command after the third try, it may either quit or send an <code>Identify</code> command to achieve limited use of iAP. See The 3G iPod (page 595) and "Interfacing With the 3G iPod" in <i>MFi Accessory Hardware Specification</i> .			
C		GetAccessoryInfo	the accessory ignores this query
The accessory is now assured that it can successfully identify its lingo to the attached Apple device.			
1	IdentifyDevice-Lingoes		identifying lingo 'General Simple Remote Display Remote Digital Audio'; options 'auth:immediate; power:low; device ID 0x00000200
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
3		GetDevAuthentication-Info	no params
4	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1;
5		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
6	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1;

Step	Accessory command	Apple device command	Comment
7		AckDevAuthentication-Info	acknowledging 'auth info supported'
8		GetAccSampleRateCaps	no params
9		GetAccessoryInfo	requesting 'Acc info capabilities'
10		GetDevAuthentication-Signature	offering challenge
11	RetAccSampleRateCaps		returning sample rates '8000 11025 12000 16000 22050 24000 32000 44100 48000'
12	RetAccessoryInfo		returning 'Acc info capabilities Acc name Acc FW version Acc HW version Acc manufacturer Acc model number Acc serial number Acc incoming max packet size' in response to 'Acc info capabilities'
13	RetDevAuthentication-Signature		returning signature
14		GetAccessoryInfo	requesting 'Acc name'
15	RetAccessoryInfo		returning 'Apple' in response to 'Acc name'
16		TrackNewAudio-Attributes	sample rate 44100; Sound Check value 0; track volume adjustment 0
17	AccAck		acknowledging 'Success (OK)' to command 'Digital Audio Lingo::TrackNewAudioAttributes'
18		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
19		GetAccessoryInfo	requesting 'Acc FW version'
20	RetAccessoryInfo		returning 'v9.8.7' in response to 'Acc FW version'
21		GetAccessoryInfo	requesting 'Acc HW version'
22	RetAccessoryInfo		returning 'v1.2.3' in response to 'Acc HW version'
23		GetAccessoryInfo	requesting 'Acc manufacturer'
24	RetAccessoryInfo		returning 'Apple Inc.' in response to 'Acc manufacturer'

Step	Accessory command	Apple device command	Comment
25		GetAccessoryInfo	requesting 'Acc model number'
26	RetAccessoryInfo		returning 'ModelNumber-XYZ' in response to 'Acc model number'
27		GetAccessoryInfo	requesting 'Acc serial number'
28	RetAccessoryInfo		returning 'SerialNumber-1234' in response to 'Acc serial number'
29		GetAccessoryInfo	requesting 'Acc incoming max packet size'
30	RetAccessoryInfo		returning '1024 bytes' in response to 'Acc incoming max packet size'

Table F-46 Non-IDPS radio tagging command sequence

Step	Accessory command	Apple device command	Comment
Steps A–C cancel any current authentication process; see "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). Step B also detects if the attached Apple device is a 3G iPod.			
A	IdentifyDevice-Lingoes		identifying lingo 'General'; options none; device ID 0x00000000
The accessory waits up to 1 second for the Apple device to respond.			
B		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
If the Apple device does not send an ACK command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an ACK command after the third try, it may either quit or send an <code>Identify</code> command to achieve limited use of iAP. See The 3G iPod (page 595) and "Interfacing With the 3G iPod" in <i>MFi Accessory Hardware Specification</i> .			
C		GetAccessoryInfo	the accessory ignores this query
The accessory is now assured that it can successfully identify its lingoes to the attached Apple device.			
1	IdentifyDevice-Lingoes		identifying lingoes 'General'; options 'auth:none; power:low'; device ID 0x00000000
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
3	GetiPodOptions		no params

Step	Accessory command	Apple device command	Comment
4		RetiPodOptions	returning 'video output line out'
5	RequestLingoProtocol-Version		requesting General Lingo version
6		ReturnLingoProtocol-Version	returning General Lingo v1.09
7	RequestLingoProtocol-Version		requesting Storage Lingo version
8		ReturnLingoProtocol-Version	returning Storage Lingo v1.02
9	IdentifyDevice-Lingoes		identifying lingoes 'General Storage'; options 'auth:immediate; power:low'; device ID 0x00000200
10		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
11		GetDevAuthentication-Info	no params
12	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1; cert data: ...
13		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
14	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1; cert data: ...
15		AckDevAuthentication-Info	acknowledging 'auth info supported'
16		GetAccessoryInfo	requesting 'Acc info capabilities'
17		GetDevAuthentication-Signature	offering challenge '...' with retry counter 1
18	RetDevAuthentication-Signature		returning signature '...'
19		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'

Step	Accessory command	Apple device command	Comment
20	RetAccessoryInfo		returning 'Acc info capabilities Acc name Acc FW version Acc HW version Acc manufacturer Acc model number Acc incoming max packet size' in response to 'Acc info capabilities'
21		GetAccessoryInfo	requesting 'Acc name'
22	RetAccessoryInfo		returning 'Radio' in response to 'Acc name'
23		GetAccessoryInfo	requesting 'Acc FW version'
24	RetAccessoryInfo		returning 'v1.0.0' in response to 'Acc FW version'
25		GetAccessoryInfo	requesting 'Acc HW version'
26	RetAccessoryInfo		returning 'v1.0.0' in response to 'Acc HW version'
27		GetAccessoryInfo	requesting 'Acc manufacturer'
28	RetAccessoryInfo		returning 'Radio Manufacturer' in response to 'Acc manufacturer'
29		GetAccessoryInfo	requesting 'Acc model number'
30	RetAccessoryInfo		returning 'M78901LL/Z' in response to 'Acc model number'
31		GetAccessoryInfo	requesting 'Acc incoming max packet size'
32	RetAccessoryInfo		returning '2048 bytes' in response to 'Acc incoming max packet size'
33	GetiPodCaps		no params
34		RetiPodCaps	returning 'total space 2147483648; max file size 1073741824; max write size 500; v1.02'
35	GetiPodFreeSpace		no params
36		RetiPodFreeSpace	returning 130023424 bytes
37	OpeniPodFeatureFile		opening feature type 'Radio Tagging'
38		RetiPodFileHandle	returning file handle 0

Step	Accessory command	Apple device command	Comment
39		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
40	CloseiPodFile		closing file with handle 0
41		iPodAck	acknowledging 'Success' to command 'Storage Lingo::CloseiPodFile' with file handle 0

Table F-47 Non-IDPS cardio equipment command sequence

Step	Accessory command	Apple device command	Comment
Steps A–C cancel any current authentication process; see "Cancelling a Current Authentication Process With IdentifyDeviceLingoes" (page 99). Step B also detects if the attached Apple device is a 3G iPod.			
A	IdentifyDevice-Lingoes		identifying lingo 'General'; options none; device ID 0x00000000
The accessory waits up to 1 second for the Apple device to respond.			
B		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
If the Apple device does not send an ACK command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an ACK command after the third try, it may either quit or send an <code>Identify</code> command to achieve limited use of iAP. See The 3G iPod (page 595) and "Interfacing With the 3G iPod" in <i>MFi Accessory Hardware Specification</i> .			
C		GetAccessoryInfo	the accessory ignores this query
The accessory is now assured that it can successfully identify its lingoes to the attached Apple device.			
1	IdentifyDevice-Lingoes		identifying lingoes 'General'; options 'auth:none; power:low'; device ID 0x00000000
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
3	GetiPodOptions		no params
4		RetiPodOptions	returning 'video output line out'
5	RequestLingoProtocol-Version		requesting General Lingo version

Step	Accessory command	Apple device command	Comment
6		ReturnLingoProtocol-Version	returning General Lingo v1.09
7	RequestLingoProtocol-Version		requesting Sports Lingo version
8		ReturnLingoProtocol-Version	returning Sports Lingo v1.01
9	RequestLingoProtocol-Version		requesting Storage Lingo version
10		ReturnLingoProtocol-Version	returning Storage Lingo v1.02
11	IdentifyDevice-Lingoes		identifying lingoes 'General Sports Storage'; options 'auth:immediate; power:low'; device ID 0x00000200
12		ACK	acknowledging 'Success (OK)' to command 'General Lingo:: IdentifyDeviceLingoes'
13		GetDevAuthentication-Info	no params
14	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1; cert data: ...
15		ACK	acknowledging 'Success (OK)' to command 'General Lingo:: RetDevAuthenticationInfo'
16	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1; cert data: ...
17		AckDevAuthentication-Info	acknowledging 'auth info supported'
18		GetDeviceCaps	no params
19		GetAccessoryInfo	requesting 'Acc info capabilities'
20		GetDevAuthentication-Signature	offering challenge '...' with retry counter 1
21	RetDevAuthentication-Signature		returning signature '...'
22		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'

Step	Accessory command	Apple device command	Comment
23		GetDeviceCaps	no params
24	RetDeviceCaps		returning 'Gym equipment command support' with max 0 node filters
25		GetDeviceCaps	no params
26	RetDeviceCaps		returning 'total space 0; max file size 0; max write size 0; read-only; no subdirs; random writes allowed; max file count 0; max name length 0; file system type Reserved; v1.02'
27	RetAccessoryInfo		returning 'Acc info capabilities Acc name Acc FW version Acc HW version Acc manufacturer Acc model number Acc incoming max packet size' in response to 'Acc info capabilities'
28		GetAccessoryInfo	requesting 'Acc name'
29	RetAccessoryInfo		returning 'Bike' in response to 'Acc name'
30		GetAccessoryInfo	requesting 'Acc FW version'
31	RetAccessoryInfo		returning 'v1.0.0' in response to 'Acc FW version'
32		GetAccessoryInfo	requesting 'Acc HW version'
33	RetAccessoryInfo		returning 'v1.0.0' in response to 'Acc HW version'
34		GetAccessoryInfo	requesting 'Acc manufacturer'
35	RetAccessoryInfo		returning 'Bike Manufacturer' in response to 'Acc manufacturer'
36		GetAccessoryInfo	requesting 'Acc model number'
37	RetAccessoryInfo		returning 'M78901LL/Z' in response to 'Acc model number'
38		GetAccessoryInfo	requesting 'Acc incoming max packet size'
39	RetAccessoryInfo		returning '2048 bytes' in response to 'Acc incoming max packet size'
40	GetiPodCaps		no params

Step	Accessory command	Apple device command	Comment
41		RetiPodCaps	returning 'Gym equipment support User data support' with userCount of 1
42	GetiPodCaps		no params
43		RetiPodCaps	returning 'total space 2147483648; max file size 1073741824; max write size 500; read-only; no subdirs; random writes allowed; max file count 10000; max name length 200; file system type HFS+; v1.02'
44	GetUserIndex		no params
45		RetUserIndex	returning userIndex of 0
46	GetUserData		requesting 'Preferred unit system'
47		RetUserData	returning 'No information' for data type 'Preferred unit system'
48	GetUserData		requesting 'Name'
49		RetUserData	returning 'NewUser' for data type 'Name'
50	GetUserData		requesting 'Gender'
51		RetUserData	returning 'No information' for data type 'Gender'
52	GetUserData		requesting 'Weight'
53		RetUserData	returning '92.0 kg' for data type 'Weight'
54	GetUserData		requesting 'Age'
55		RetUserData	returning '26 years' for data type 'Age'
56	GetUserData		requesting 'Recording preference'
57		RetUserData	returning 'No information' for data type 'Recording preference'
58	OpeniPodFeatureFile		opening feature type 'Gym Equipment Workout' with options mask 'file data ipodInfo XML signature' and file data:</gymData>;
59		RetiPodFileHandle	returning file handle 0

Step	Accessory command	Apple device command	Comment
60	WriteiPodFileData		writing 293 bytes at offset 0 and handle 0: <?xml version=""1.0"" encoding=""UTF-8""?>; <gymData>; <vers>1</vers>; <equipmentInfo>; <manufacturerID>00000001</manufacturerID>; <manufacturerName>Bike Manufacturer</manufacturerName>; <type>Bike</type>; <model>M78901LL/Z</model>; <serialNumber> UV1234567890-SN</serialNumber>; </equipmentInfo>;
61		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
62	WriteiPodFileData		writing 125 bytes at offset 240 and handle 0: <userInfo><kg>92.0</kg></userInfo>; <template>; <templateName>Athletic Challenge</templateName>; <sec>60</sec>; </template>;
63		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
64	WriteiPodFileData		writing 125 bytes at offset 366 and handle 0: <interval>; <event>start</event>; <sec>0</sec>; <kCal>0</kCal>; <km>0.00</km>; <rpm>26</rpm>; <level>1</level>; </interval>;
65		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
66	WriteiPodFileData		writing 86 bytes at offset 392 and handle 0: <interval>; <sec>10</sec>; <kCal>0</kCal>; <km>0.02</km>; <rpm>64</rpm>; </interval>;
67		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
68	WriteiPodFileData		writing 87 bytes at offset 479 and handle 0: <interval>; <sec>20</sec>; <kCal>12</kCal>; <km>0.7</km>; <rpm>189</rpm>; </interval>;

Step	Accessory command	Apple device command	Comment
69		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
70	WriteiPodFileData		writing 107 bytes at offset 567 and handle 0: <interval>; <event>end</event>; <sec>21</sec>; <kCal>13</kCal>; <km>0.8</km>; <rpm>170</rpm>; </interval>;
71		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
72	WriteiPodFileData		writing 99 bytes at offset 675 and handle 0: <workoutSummary>; <sec>21</sec>; <kCal>13</kCal>; <km>0.8</km>; <rpm>170</rpm>; </workoutSummary>;
73		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
74	CloseiPodFile		closing file with handle 0
75		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: CloseiPodFile' with file handle 0

Glossary

accessory A third-party device licensed under the Made for iPod program.

authentication A mechanism used by an Apple device to verify whether an attached accessory is an authorized accessory and by an accessory to authenticate the Apple device, if desired.

checksum The byte sum of packet bytes from the payload length through the last packet byte. This is used to validate the contents of a command packet. For a valid packet, the sum of the bytes, including the checksum byte, must be 0x00. The packet checksum byte—the last byte in a packet—must be the 2's complement (the negative) of the sum of the payload length byte up to, but not including, the packet checksum byte.

deprecated Used to describe a technology or feature that is supported but whose use is discouraged and not recommended. Such a technology or feature has typically been replaced by a newer one and is likely to become unsupported in the future.

FID A Full ID string, used by IDPS to send token-value fields that an Apple device is able to parse during the accessory identification process.

HID (Human Interface Device) HID is a standard USB class. A USB host such as a PC or Macintosh will recognize any attached USB device that supports a HID interface and makes it available to the application layers of the operating system via a set of programming interfaces. A common application of a HID interface is a USB mouse or joystick.

HID report A single unit of data that is used to send information to the HID interface of an Apple device or from the Apple device to the host. iAP packets are broken into HID reports before being sent across the USB port link and are reassembled on the receiving side.

IDPS Identify Device Preferences and Settings, the identification process required for an accessory to communicate with an Apple device. See "[Accessory Identification](#)" (page 519).

iUI (iPod USB Interface) A configuration of an Apple device when attached as a device over USB. This configuration allows the Apple device to be controlled using iAP, using a USB HID class interface as a transport mechanism.

LCB (Link Control Byte) A byte used by the iUI to indicate report sets and manage data flow.

lingo The command category used by an accessory. There is a General lingo that must be supported by all accessories. Other lingo's are designed for use by specific accessories, such as simple remote controls and microphones.

link The logical connection between an external accessory and an Apple device via serial port or other physical connection.

low power mode An operating mode of an accessory in which it draws low power from an attached Apple device, as defined in the section "Accessory Power Policy" in *MFi Accessory Hardware Specification*.

packet The logical set of bytes that compose a valid command sequence. This set includes the packet start byte, packet payload length, payload, and payload checksum. Note that a sync byte is appended to the beginning of the packet when using the UART serial port as the data transport link. There are two different packet types: small format and large format.

payload The sequence of bytes consisting of the lingo, command, and data that are contained within a packet.

podcasting A way to publish multimedia files on the Internet that lets users receive new files automatically by subscription. Podcast files are typically downloaded to Apple devices through Apple's iTunes application.

RDS/RBDS (Radio [Broadcast] Display System) A technology for broadcasting and displaying artist, album, track titles, and similar information on FM radio receivers.

resistor-based accessory An accessory that uses an Accessory Identify resistor to access only limited functions in an Apple device. Compare Serial accessory.

RSSI (Receive Signal Strength Indicator) A measure of the strength of an RF signal coming into a radio frequency tuner.

serial accessory An accessory that uses the Accessory Protocol Interface to access a range of Apple device functions. Compare Resistor-based accessory.

UART (Universal Asynchronous Receiver/Transmitter) A piece of computer hardware that translates between parallel and serial bits of data. A UART is usually an integrated circuit used for serial communications over a computer or peripheral accessory serial port.

USB (Universal Serial Bus) An interface standard for communication between a computer and external peripherals over a cable using biserial transmission.

USB descriptor A standard USB data structure that is passed from a USB device to the host upon request. Descriptors are used by the USB device to communicate its characteristics and resource requirements to the host.

USB endpoint A logical connection point that is used to set up a data transfer pipe between a USB host and interface on an accessory. For instance, the HID interface on Apple devices uses an interrupt-type endpoint to enable a pipe for transferring data to the USB Host.

USB host A single computer connected to one or more USB devices or functions. The host is responsible for recognizing that a USB device has been attached to it and for driving the communications with the device. For the purposes of this document, an Apple device is a USB device that provides a function, and

an accessory is the USB host. However, an Apple device can also be made a USB host with the accessory acting as its USB device.

X.509 certificate A standard defined by the International Telecommunication Union (ITU) that governs the format of certificates used for authentication and sender identity verification in public-key cryptography. In the iAP, X.509 certificates contain the public keys used in the authentication process.

Document Revision History

This table describes the changes to *MFi Accessory Firmware Specification*.

Date	Notes
2011-04-04	Revision R43: Updated for iPad 2, iPhone 4 (CDMA model), and iOS 4.3.1.
	Updated "Notice of Proprietary Property" (page 31).
	Updated General lingo ACK command formats; see Table B-1 (page 526).
	Added General lingo bit 30 to RetiPodOptionsForLingo values; see Table 2-138 (page 167).
	Deprecated and removed General lingo commands 0x0D, RequestiPodModelNum, and 0x0E, ReturniPodModelNum.
	Changed names of General lingo commands RequestTransportMaxPacketSize and ReturnTransportMaxPacketSize to RequestTransportMaxPayloadSize and ReturnTransportMaxPayloadSize.
	Incorporated numerous other updates and corrections.
2010-11-29	Revision R42: Updated for iOS 4.2.1.
	Corrected generic references to Apple devices, iOS devices, and iPods throughout the document; see "IMPORTANT" (page 31).
	Globally replaced SDKProtocolToken and SDKProtocolMetadataToken with EAPProtocolToken and EAPProtocolMetadataToken.
	Added new section "iPod Out Mode" (page 59).
	Added new section "Upgrading from UART Transport to USB Host Mode" (page 55).
	Updated section "Applicability of Transaction IDs" (page 526).
	Updated section "Command 0x0028: PlayCurrentSelection" (page 475).
	Restored documentation of USB Host mode power management command "Command 0x54: SetAvailableCurrent" (page 180).
	Restored documentation of USB Host mode charging notifications; see Table 2-133 (page 164).

Date	Notes
	Reserved Accessory Info Type 0x02; see Table 2-51 (page 112) and Table 2-55 (page 114).
	Deprecated the Microphone lingo; see "Deprecated Lingo 0x01: Microphone Lingo" (page 600).
	Deprecated "Command 0x0038: SelectSortDBRecord" (page 618).
	Deprecated chapter play control commands listed in Table 4-60 (page 477).
	Removed section "Accessory Power Policy" from Chapter 2 and inserted a reference to the same section of <i>MFi Accessory Hardware Specification</i> in section "Lingo 0x05: Accessory Power Lingo" (page 271).
2010-09-08	<i>Revision R41:</i> Updated for 6G nano, 4G touch, and iOS 4.1.
	Added Display Remote lingo commands 0x21 and 0x22 to support Genius playlists; see Table 3-50 (page 227).
	Added Extended Interface lingo commands 0x44-0x45 and 0x47-0x49 to support Genius playlists; see Table 4-10 (page 436).
	Added new section "Playback Status Notifications" (page 418).
	Added new section "Applicability of Transaction IDs" (page 526).
	Added new sections "Audio and Video Output Preferences" (page 47) and "Audio/Video Synchronization."
	Documented new USB HID commands (see Table 3-12 (page 197)).
	Deprecated iPod alarm events; see Table 3-62 (page 236).
	Changed name of Digital Audio lingo "Command 0x04" (page 360) from <code>NewiPodTrackInfo</code> to <code>TrackNewAudioAttributes</code> .
	Noted that <code>SelectSortDBRecord</code> is not supported on iPods that run iOS.
	Added transaction ID fields to <code>GetDevAuthenticationSignature</code> and <code>RetDevAuthenticationSignature</code> packets.
	Corrected payload size range of large iAP packets; see "Large Packet Format" (page 77).
	Revised General lingo command 0x0E, <code>ReturniPodModelNum</code> , to show that it no longer returns meaningful model IDs.
	Clarified generic references to iPods and iOS devices; see "Note" (page 31).
2010-06-24	<i>Revision R40:</i> Updated for iPhone 4.
	Changed document name from "iPod Accessory Protocol Interface Specification" to "MFi Accessory Firmware Specification."

Date	Notes
	Changed name of "iPhone OS" to "iOS" for version 4 and later versions.
	Added information for 3rd generation iPod touch.
	Added section "Lingo 0x0D: iPod Out Lingo" (page 374).
	Added section "Accessory Launching of iOS Applications" (page 49).
	Added section "USB Human Interface Device Reports" (page 197).
	Added section "User Interface Accessibility Commands" (page 198).
	Added section "Controlling Applications Using the Simple Remote Lingo" (page 53).
	Updated section "Using an Apple Device as a USB Host" (page 54).
	Added General lingo "Command 0x11: RequestTransportMaxPacketSize" (page 95) and "Command 0x12: ReturnTransportMaxPacketSize" (page 95).
	Added General lingo "Command 0x50: CancelCommand" (page 178) for multipacket data transfers.
	Added new notifications; see "Command 0x4A: iPodNotification" (page 159).
	Documented <code>SessionWriteFailure</code> error for large data transfers; see Table 2-17 (page 87).
	Documented accessory RF certification declarations; see Table 2-83 (page 136).
	Documented application matching controls; see Table 2-88 (page 137).
	Documented <code>ScreenInfoToken</code> for iPod Out mode; see Table 2-89 (page 138).
	Deprecated Simple Remote lingo "Command 0x02: ImageButtonStatus" (page 611).
	Incorporated numerous other updates and corrections.
2010-04-09	<i>Revision R39:</i> Updated for iPad.
	Added new appendix A, "iAP Interfaces for the iPad".
	Added USB Host mode power management commands 0x52–0x54, 0x56, and 0x58–0x59 to the General lingo, as listed in Table 2-9 (page 79).
	Added commands 0x46–0x48 to the General lingo to support accessory status notifications, as listed in Table 2-9 (page 79), and added Status Info type 0x0B to <code>GetAccessoryInfo</code> and <code>RetAccessoryInfo</code> .
	Restructured Appendix E, "iTunes Tagging" (page 535), and updated it for additional broadcast sources.
	Incorporated numerous updates and corrections.

Date	Notes
2009-10-22	<i>Revision R38:</i>
	Added section "Reserved Commands and Data" (page 65).
	Added information for the 5G nano, the 2G touch (2009), and the iPod classic 160 GB.
	Added commands 0x49-0x4A, 0x4D-0x4F, and 0x51 to the General lingo to support Event Notifications; see Table 2-9 (page 79). Also added explanatory section "Apple Device Event Notifications" (page 523).
	Added commands 0x0D and 0x0E to the Simple Remote lingo to support button actions for 5G nano Radio Tagging and Camera accessories; see Table 3-3 (page 185).
	Added section "Accessory Control of the iPod 5G nano Camera" (page 190).
	Added commands 0x25-0x31 to the RF Tuner lingo, and added features to other commands, to support HD radio; see Table 3-121 (page 280).
	Incorporated numerous updates and corrections.
2009-09-09	<i>Revision R37:</i>
	Exported the following chapters and appendixes to new book <i>iPod/iPhone Hardware Specifications</i> , Release R1: "Hardware Interfaces," "Functional Description," "Protocol Transport Links," "iPod Power States and Accessory Power," "Headphone Remote and Mic System," "Interfacing With the 3G iPod," "Sample Accessory Circuits," "Power Guidelines," and "FireWire to USB Reference Design."
	Exported appendix "Headset and FireWire-to-USB Power Converter Certification" to new book <i>iPod/iPhone Accessory Testing and Certification Specification</i> , Release R1.
	Imported entire contents of discontinued book <i>iPod Extended Interface Specification</i> , Release R25, into "The Extended Interface Protocol" (page 409).
	Imported some content from discontinued book <i>iPhone Accessory Interface Specification</i> , Release R9.
	Reformatted content to new layout and typography.
2009-06-23	<i>Revision R36:</i>
	Added information about the iPhone 3GS.
	Added new appendix "Accessory Identification" (page 519). <i>This identification process is required in all new accessory designs.</i> Moved command examples using IdentifyDeviceLingoes to "Accessory Identification With Non-IDPS Apple Devices" (page 620) in Appendix M, "Historical Information."

Date	Notes
	In Chapter 3, added new section "Accessory Communication With iOS Applications" (page 50); deleted section "iPod Games" and updated section "Accessory Control of iOS Devices" (page 49).
	Added commands 0x38-0x3C, 0x3F-0x43, and 0x4A-0x4C to the General lingo; see Table 2-9 (page 79).
	Added new section "Accessory Power Policy."
	Added new section "Lingo 0x0E: Location Lingo" (page 380).
	Added sample command sequences for iTunes tagging (Table D-6 (page 546)) and cardio equipment (Table E-7 (page 584)).
	Added new appendix "Transaction IDs" (page 525).
	Added new appendix "Multisection Data Transfers" (page 531).
	Added new section "Avoiding an iPhone OS Warning When a Self-Powered Accessory Is Off."
	Added caution about latching connectors to "30-Pin Connector."
	Changed button press detection parameters in "iPod/iPhone Headphone/Microphone Jack."
	Added information about PKCS-7 transport of certificate data to "Accessory Authentication of the Apple Device" (page 75).
	Added warning about the iPod touch and iPhone Off state to "Power States."
	Updated signal level information in "UART Serial Port Link."
	Added new section "Examples of Transaction IDs" (page 527).
	Added new section Sample Identification Sequences (page 402).
2009-03-17	<i>Revision R35:</i>
	Added Table I-1 (page 32) to identify iPod/iPhone models.
	Revised "iPhone headphone/microphone schematic" to show iPod/iPhone differences.
	Revised section "Line Level Input" in Chapter 2.
	Updated Table 1-3 (page 43) to show current firmware versions.
	Added display resolution data to Table 1-8 (page 48).
	Updated and revised accessory identification and authentication in Table 2-1 (page 67) and Table F-40 (page 620).
	Added new ACK error codes to Table 2-14 (page 85).

Date	Notes
	Added copy protection requirement to "USB Audio Transport" (page 350).
	Revised section "Button Detection Circuitry" in Appendix C.
2009-01-05	<i>Revision R34:</i>
	Added definitions of legal agreement terminology in Introduction.
	Added new appendix "Accessory Certification."
	Added FM radio tagging information to "iTunes Tagging" (page 535).
	Added new section "Lingo 0x09: Sports Lingo" (page 336).
	Added new appendix "Nike + iPod Cardio Equipment System" (page 569).
	Added new appendix "Headphone Remote and Mic System."
	Added new commands 0x80-0x82 and new options for command 0x12 in "Lingo 0x0C: Storage Lingo" (page 362).
	Documented new models: the 4G iPod nano, 120 GB classic, and 2G iPod touch.
	Added section "Accessory Control of iOS Devices" (page 49).
	Added hardware details of the iPhone audio connection ("iPod/iPhone Headphone/Microphone Jack").
	Clarified USB power requirements in "USB 2.0."
	Added section "Magnetic Sensitivity of the iPod."
	Defined recommended procedure for determining iPod capabilities in "Command 0x13: IdentifyDeviceLingoes" (page 96).
	Added section "Playback Engine Playlists" (page 188).
	Clarified behavior of Next and Previous buttons in "Using Contextual Buttons" (page 188).
	Clarified mute event in Table 3-64 (page 238).
	Deprecated commands in "Deprecated USB Host Control Commands" (page 613).
	Deprecated "General Lingo Command 0x01: Identify" (page 598)).
	Removed from Chapter 5 obsolete description of testing for the General lingo over the UART serial port link.
2008-06-26	<i>Revision R33:</i>
	Added section "Command Timings" (page 184).

Date	Notes
	Revised audiobook playback speeds in Table 3-69 (page 244).
	Added accessory requirements to Table 2-57 (page 115).
	Changed names of levels of authentication from V1 and V2 to authentication 1.0 and 2.0.
	Changed name of RF Transmitter lingo (0x05) to Accessory Power lingo.
	Added section "Supplying USB Power" to Appendix A.
	Changed " Data Transfer to the Apple Device " (page 541) in Appendix B to show all radio tagging plist fields required.
	Added section "Powering the 3G iPod" to Appendix C.
	Added Note about lyrics strings to Table 3-86 (page 258).
	Revised "Typical diode bridge circuit for an AC adapter" in Appendix E.
	Fixed typo in Table F-2 in Appendix F.
2008-05-07	<i>Revision R32:</i>
	Added new appendix, "FireWire to USB Reference Design."
	Revised documentation of sleep states in Chapter 2, Appendix A, and elsewhere.
	Added section " Video Output Settings " (page 48) and expanded Table 2-66 (page 120).
	Updated the current firmware versions in Table 1-2 (page 42).
	Made several updates to Appendix B, "iTunes Tagging."
	Made numerous other updates, corrections, and clarifications throughout the document.
2007-12-12	<i>Revision R31:</i>
	Moved past firmware version documentation, description of the 9-pin Audio/Remote connector, FireWire specifications, and other noncurrent material to new appendix, " Historical Information " (page 591).
	Documented November 2007 firmware for the 5G, classic, 3G nano, and touch iPod models.
	Made minor correction to "Configuration and interface descriptors for iPods with USB audio."
	Simplified document structure.
2007-10-02	<i>Revision R30:</i>

Date	Notes
	Added " Lingo 0x0C: Storage Lingo " (page 362) to Chapter 6.
	Added new Appendix B, " iTunes Tagging " (page 535).
2007-09-05	<i>Revision R29:</i>
	Added documentation for the iPod classic, iPod 3G nano, and iPod touch.
	Added documentation for component video outputs.
	Added new command, <code>SetVideoDelay</code> , to the Digital Audio lingo.
	Deprecated the FireWire interface on the 30-pin connector.
	Added new preference IDs to <code>GetiPodPreferences</code> .
	Added design guidelines for third-party developers of AC adapter accessories for the iPod touch ("Power Guidelines").
	Added design guidelines for third-party developers of carrying cases for iPod touch ("iPod touch Carrying Case Design").
2007-06-29	<i>Revision R28:</i>
	Revised pin connections in 30-pin to FireWire cable ("30-pin to FireWire cable").
	Updated model listings to include iPhone and new iPod models.
	Updated requirements for artwork count data (Table 3-86 (page 258))
	Added caution about sending signals to the iPod UART when its serial receive block is off.
	Documented X.509 certificate classes (Table 2-3 (page 72))
	Added line-out usage controls (Table 2-66 (page 120))
	Added section " USB Audio Errors on Older Apple Devices " (page 355)
	Added authentication requirement to General lingo commands 0x1A-0x1F
	Deprecated Level V1 authentication for new designs (see " Apple Device Authentication of the Accessory " (page 73))
	Deprecated " Command 0x01: Identify " (page 598)
2007-02-06	<i>Revision R27:</i>
	Added section "Minimizing Crosstalk and Noise."
	Added new information to Table 1-2 (page 42).
	Removed autobaud on parity errors from Table 1-5 (page 44).

Date	Notes
	Clarified UART communication rates in "UART Serial Port Link."
	Removed Manufacturer String and Product String from "Choosing an iPod USB Configuration."
	Added example of using iAP over USB in "Transferring IdentifyDeviceLingoes and ACK commands over USB using iAP."
	Distinguished behavior of audio and video playback when iPod enters Extended Interface mode in " Command 0x05: EnterRemoteUIMode " (page 89).
	Added example of using Display Remote Protocol in " Lingo 0x03: Display Remote Lingo " (page 226).
	Clarified usage of <code>NewiPodTrackInfo</code> command in " Command 0x04: TrackNewAudioAttributes " (page 360).
	Added new appendix, "Interfacing With the 3G iPod."
	Added new appendix, "Sample Accessory Circuits."
	Added temperature range to "UART Serial Port Link."
2006-11-03	Added new information to Tables 3-1 and 5-26.
2006-10-17	<i>Revision R26:</i>
	Added new information to the note after Table 2-2.
	Updated Table 2-3.
	Added note to section "Line Level Output."
	Added new section "Headphone Jack on Video-Capable iPods."
	Added new iPod models and software versions.
2006-09-12	<i>Revision R25:</i>
	Added iPod options and preferences commands (0x24-0x25 and 0x29-0x2B) to General lingo.
	Added USB Host Control lingo (0x06).
	Added RF Tuner lingo (0x07).
	Updated list of model ID strings.
	Corrected <code>RetTrackArtworkData</code> packet listing (Table 3-93 (page 263)).
2006-06-19	<i>Revision R24:</i>
	Added Accessory Equalizer lingo, number 0x08.

Date	Notes
	Added USB Digital Audio lingo, number 0x0A.
	Added Authentication level V2 (X.509 certification) to General Lingo (0x00).
	Added Album Art commands (0x16-0x19 and 0x1F-0x20) to Display Remote Lingo (0x03).
	Added GetAccessoryInfo (0x27) and RetAccessoryInfo (0x28) commands to the General lingo (0x00).
	Added Dedicated Media commands (0x00-0x04) to the Simple Remote lingo (0x02).
	Added timeout and retry information to various command descriptions.
	Moved documentation of Extended Interface commands (0x03-0x06, General lingo) from the <i>iPod Extended Interface Specification</i> .
	Revised model and feature tables in Chapter 3.
	Added and updated lingo history tables in Chapter 6.
2006-02-10	<i>Revision R23:</i>
	Pages 17 and 27: Changed audio output power specification to 25 mW.
	Page 87: A simple remote accessory must send a data payload when all buttons are released; 200 ms timeout removed.
	Corrected Table 1-2 (page 42).
	Note, page 17: Specified Technical Note TN001i by name.
2006-01-05	<i>Revision R22:</i>
	General update and reorganization of content.
	Added information on iPod power states.
	Added “D+ and D- connections for a 500 mA USB power brick” and “iPod equivalent input circuits.”
2005-10-12	Updated functional descriptions.
	Added new microphone lingo commands.
	Additional information about hibernate mode.
	Bug fixes for volume control and others.
2005-09-07	Added support for USB/iUI, authentication and additions to the new Display Remote lingo (0x03)
2005-03-21	Added Equalizer Control lingo support and commands.

Date	Notes
2004-11-12	Added general lingo commands, bug fixes and pinouts for the iPod photo release
2004-08-03	Reserved the 28k pulldown resistor
2004-07-21	Added lingo command packet examples.
	Corrected doc properties.
	Added minor clarifications.
	Added new accessory detect image.
	Updated content based on internal review.
2004-05-17	Incorporated review feedback
2004-04-20	Update simple remote, doc reformat
2004-01-14	Minor clarifications
2003-08-12	5mA access power note
2003-08-04	New serial, add bottom serial
2003-07-22	Picture to show Remote Data lines
2003-07-07	License agreement
2003-04-15	Remote protocols added
2003-04-02	Car Charger Detect added
2003-02-04	Accessory Detect Resistor Change
2003-01-09	Rx, Tx Clarification
2002-12-04	Initial Release.

