

Introducing Bluetooth[®] LE Audio (2nd edition, including Auracast[™])

A guide to the latest Bluetooth specifications and how they will change the way we design and use audio and telephony products.

2nd edition – November 2024

by Nick Hunn

First published January 2022

Second edition November 2024

Paperback ISBN: 979-8 33 795242-0

Hardback ISBN: 979-8 34 094194-7

Copyright © 2022-24 Nick Hunn

All rights reserved.

Disclaimer

The information in this book is presented for instructional value. The author has taken care in the preparation of this book, but makes no implied or expressed warranty of any kind and assumes no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages caused or alleged to be caused directly or indirectly by the information contained within it, nor for any infringements of the intellectual property rights of third parties that would result from the use of this information.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks is with permission. Other trademarks and trade names are those of their respective owners.

www.bleaudio.com | www.nickhunn.com

Acknowledgements

I would like to thank everyone who has helped towards the existence of this book. Without the work of many brilliant people contributing to the specifications, there would be nothing to write about. Trying to develop standards of this complexity results in many days and evenings of wide-ranging discussions, which have been both challenging and enjoyable. It has been a pleasure to work with so many people of passion who are always ready to spend time explaining the intricacies of wireless, audio and its application. They are all deservedly listed in the Contributor section of the Bluetooth® LE Audio specifications.

For taking the time to read through the drafts of the first edition and coming back with detailed comments, I would once more like to thank Richard Einhorn, Kanji Kerai, Ken Kolderup, Mahendra Tailor, Jonathan Tanner and Martin Woolley. Their input, from a variety of different reader angles, helped make this a much better book than it would otherwise have been. I must also thank the Bluetooth SIG for their help and support. I'd like to say a big thank you to all of the readers who have come back with suggestions and corrections in the first edition, including Mohammad Afaneh, Xavier Boniface, Bingquan Cai, Fahai Chen, Wendelin Falschlunger, Julian Fessard, Devyani Godbole, Niclas Granqvist, Pete Guerin, Jack He, Lars Knudsen, Soeren Larsen, Antoine Mettler, Jose Rodriguez Navarro, Ladislav Podivin, David Samuelson, Hai Shalom, Nick Tsai, Fei Wang, Martin Woolley and Lei Yu, along with Kevin, Matthew, Theo and Yajun. Apologies for anyone I have missed. Once again, Mahendra Tailor read much of the new text and suggested how it could be improved.

For giving me permission to use the image of Figure 1.8, and also for playing a pivotal role in kickstarting the whole world of hearables, many thanks to Nikolaj Hviid of Bragi.

Finally, to my wife, Chris, for reading the endless drafts of both editions multiple times, questioning everything and also putting up with the many hours I spent writing it.

Contents

Chapter 1.	The background and heritage.....	13
1.1	The hearing aid contribution.....	21
1.2	Limitations and proprietary extensions.....	22
1.3	What’s in a hearable?.....	27
Chapter 2.	The Bluetooth® LE Audio architecture.....	33
2.1	The use cases.....	33
2.2	The Bluetooth LE Audio architecture.....	42
2.3	Talking about Bluetooth LE Audio.....	52
Chapter 3.	New concepts in Bluetooth® LE Audio.....	56
3.1	Multi-profile by design.....	56
3.2	The Audio Sink led journey.....	56
3.3	Terminology.....	57
3.4	Context Types.....	61
3.5	The unconnected model.....	65
3.6	Availability.....	66
3.7	Announcements.....	67
3.8	Content Control ID - CCID.....	69
3.9	Coordinated Sets.....	70
3.10	Audio Location.....	71
3.11	Channel Allocation (multiplexing).....	73
3.12	Presentation Delay and serialisation of audio data.....	75
3.13	Remote controls (Commanders).....	81
Chapter 4.	Isochronous Streams.....	84
4.1	Bluetooth LE Audio topologies.....	84
4.2	Isochronous Streams and Roles.....	86
4.3	Connected Isochronous Streams.....	89
4.4	Broadcast Isochronous Streams.....	108
4.5	ISOAL – The Isochronous Adaptation Layer.....	135
Chapter 5.	LC3, latency and QoS.....	138
5.1	Introduction.....	138

5.2	Codecs and latency	139
5.3	Classic Bluetooth codecs – their strengths and limitations.....	141
5.4	The LC3 codec.....	144
5.5	LC3 latency.....	150
5.6	Quality of Service (QoS).....	151
5.7	Audio quality	158
5.8	Multi-channel LC3 audio and Audio Configurations	160
5.9	Additional codecs	165
Chapter 6.	CAP and CSIPS	168
6.1	CSIPS – the Coordinated Set Identification Profile and Service	168
6.2	CAP – the Common Audio Profile	171
Chapter 7.	Setting up Unicast Audio Streams	178
7.1	PACS – the Published Audio Capabilities Service	178
7.2	ASCS – the Audio Stream Control Service	191
7.3	BAP – the Basic Audio Profile.....	195
7.4	Stepping through the ASE and CIG configuration process.....	198
7.5	Handling missing Acceptors.....	217
7.6	Preconfiguring CISes	217
7.7	Who’s in charge?.....	218
Chapter 8.	Setting up and using Broadcast Audio Streams.....	222
8.1	Setting up a Broadcast Source	224
8.2	Starting a broadcast Audio Stream.....	226
8.3	Receiving broadcast Audio Streams	237
8.4	The broadcast reception user experience	240
8.5	Commanders as Broadcast Assistants.....	241
8.6	BASS – the Broadcast Audio Scan Service.....	242
8.7	Broadcast_Codes and encrypted broadcasts.....	251
8.8	Using Broadcast Assistants to select broadcast Audio Streams.....	251
8.9	Handovers between Broadcast and Unicast.....	262
Chapter 9.	Telephony and Media Control	264
9.1	Terminology and Generic TBS and MCS features.....	265
9.2	Control topologies.....	267

9.3	TBS and CCP	268
9.4	MCS and MCP	276
Chapter 10.	Volume, Audio Input and Microphone Control.....	284
10.1	Volume and input control.....	284
10.2	Volume Control Service	286
10.3	Volume Offset Control Service.....	289
10.4	Audio Input Control Service	290
10.5	Putting the volume controls together.....	294
10.6	Mixing and real world implementations	295
10.7	Microphone control	296
10.8	Local volume control.....	298
10.9	A codicil on terminology and multiple Volume Controllers	299
Chapter 11.	Top level Bluetooth® LE Audio profiles and the Broadcast Audio URI ...	302
11.1	HAPS - the Hearing Access Profile and Service	303
11.2	TMAP – The Telephony and Media Audio Profile	307
11.3	GMAP – the Gaming Audio Profile	311
11.4	Broadcast Gaming Roles – BGS and BGR.....	314
11.5	GMAP synchronization.....	315
11.6	GMAP features and GMAS.....	315
11.7	Interoperability between HAP, TMAP and GMAP	316
11.8	Public Broadcast Profile	318
11.9	The Broadcast Audio URI (BAU).....	321
Chapter 12.	Auracast™	326
12.1	The basic principles	327
12.2	The Auracast™ Simple Transmitter Best Practices Guide.....	331
12.3	The Broadcast Assistant	334
12.4	Auracast™ Assistant and Receiver Guides	336
12.5	Virtual Broadcast Assistants	341
Chapter 13.	Practical considerations	344
13.1	Latency and Presentation Delay in real implementations	344
13.2	Mixing Broadcast and Unicast.....	350
13.3	Using broadcast with TVs.....	361

13.4	The third device – The Broadcast Assistant.....	367
Chapter 14.	Bluetooth® LE Audio applications.....	372
14.1	Changing the way we acquire and consume audio.....	372
14.2	Broadcast for all.....	374
14.3	TVs and broadcast.....	379
14.4	Managed Audio Services	381
14.5	Phones and broadcast.....	381
14.6	Microphones.....	383
14.7	Personal communication.....	384
14.8	The future of earbuds	386
14.9	Market development and notes for developers	388
Chapter 15.	Glossary and concordances	392
15.1	Abbreviations and initialisms.....	392
15.2	Bluetooth LE Audio specifications	396
15.3	Procedures in Bluetooth LE Audio.....	397
15.4	Bluetooth LE Audio characteristics.....	401
15.5	Bluetooth LE Audio terms	403

Introduction to the second edition

When I wrote the first edition of this book, we were still finalising the Bluetooth® LE Audio specifications. Although many of the companies involved had prototypes that allowed us to test the features of the specification, nobody had made any real products, so many aspects of the specification were still largely theoretical. Since then, developers have been working hard incorporating the Bluetooth LE Audio specifications into products. As I have been writing this edition, the first few products have been launched on the market.

What those intervening months have taught us is just how different the architecture of Bluetooth LE Audio is to what had come before. In particular, it has highlighted the fact that there is more work to do in explaining the differences and helping developers to understand the potential of these new specifications. Several areas stand out, which I have tried to address in this second edition.

The first of these is broadcast, and particularly the role of the Broadcast Assistant. I've rewritten much of Chapter 8 to try and provide a clearer explanation of how to use these features and turn them into real products. Since publishing the first edition, the Public Broadcast Profile and Broadcast Audio URI specification have been adopted, but perhaps the most important development is the Bluetooth SIG's promotion of the Auracast™ brand for public and personal broadcast experiences, which includes detailed guidelines of how these devices should operate. That is now covered in a new Chapter 12.

At a more basic architectural level, many implementers have failed to notice the ramifications of the change from the constantly connected model that evolved with A2DP and HFP, where a device that makes a connection tends to keep it for life, to the approach of the Bluetooth LE Audio specifications, where a connection is only maintained for as long as a use case requires it. That flexibility was added to help prevent a proliferation of multi-profile issues, which have hindered classic Bluetooth audio from expanding into new use cases. I've added Chapter 13, which looks at some of the practical issues that have been seen in some of the early product iterations. I hope that it helps explain those differences and how they can be addressed.

Another example of implementers following what they know from classic has been the continued use of stereo streams, where both left and right encoded packets are sent to every device, rather than the more efficient method of using a CIS or BIS to send only the audio information that a device requires. I've tried to provide more clarity on that.

The concept of Audio Locations has also caused confusion, and the BAP and PACS specifications have just been updated to provide clarity on the use of mono signals. I've expanded their description in several sections to reflect that and make it clearer how to use this feature.

As is inevitable in such a large set of specifications, there have been items where we've needed to address mistakes and omissions. There have been relatively few changes which affect implementation, but where they are significant, I've tried to highlight them in the relevant sections.

There have also been a few more specifications added to the Bluetooth LE Audio world. In chapter 11, I've added the Gaming Audio Profile (GMAP) and the Broadcast Audio URI (BAU). I've also updated the final chapter to provide a current perspective of how Bluetooth LE Audio is likely to change the way that everyone uses audio. The use cases are still evolving and there will be more we've not yet anticipated. Bluetooth LE Audio gives developers the opportunity to think up totally new ways to use and share audio. It should be an exciting future.

November 2024

Introduction to the first edition

Back in the spring of 2013, I remember sitting in a conference room in Trondheim with representatives of the hearing aid industry as they explained to the Bluetooth Board of Directors why they should commit time and effort to develop a Bluetooth® LE specification that would support the streaming of audio. I'd been asked by the hearing aid companies if I would chair the working group to develop the new specifications. Everyone agreed it was a good idea and the two groups – the Bluetooth Special Interest Group (SIG), and EHIMA – the Hearing Instrument Manufacturer's Association – the trade body representing the industry, signed a Memorandum of Understanding to start work on a new specification to support audio over Bluetooth LE.

At the time, we all thought it would be a fairly quick development – hearing aids didn't need enormously high audio quality – their main concern in terms of Bluetooth technology was to minimise power consumption. What none of us had realised at the time was that the technology and use cases that had been developed by the hearing aid industry were quite a long way ahead of what the consumer audio market was currently doing. Although the long-established telecoil system of inductive loops, which allowed broadcast audio to reach multiple hearing aids only provided limited quality audio, the connection topologies they supported were more complex than those provided by the existing Bluetooth A2DP and HFP audio profiles. In addition, the power management techniques and optimisations used in hearing aids gave battery lives that were an order of magnitude greater than those in similarly sized consumer products.

Over the next twelve months, as we developed functional requirements documents, more and more of the traditional audio and silicon companies came to look at what we were doing, and decided that many of the features that we were proposing for hearing aids were equally applicable to their markets. In fact, they appeared to solve many of the limitations that existed in the current Bluetooth audio specifications. As a result, the requirements list grew and the small hearing aid project evolved into the largest single specification development that the Bluetooth SIG has ever done, culminating in what is now collectively known as Bluetooth LE Audio.

It's hard to believe that the journey has taken eight years. At the end of it we have produced two major revisions to the Core specification, introduced a completely new, high efficiency codec and released twenty-three profile and service specifications, along with accompanying documentation and Assigned Numbers documents, which between them contain around 1,250 new pages.

For anyone not involved with that eight-year journey, it's a pretty formidable set of documents to start reading. The purpose of this book is to try and put those specifications into context, adding some of the history and rationale behind them, to help readers understand how the different parts interconnect. I've also provided some background information on the market

and what's in a hearable, to help readers relate the specification to actual products. In the chapters which delve into detail, I've included references to the specific part of the specifications using their abbreviated name and section number, e.g. [BAP 3.5.1] for Section 3.5.1 of the Basic Audio Profile. I've tried to limit the number of references, so that they don't get in the way of the text. The glossaries and concordances in Chapter 15 should also help developers navigate their way around the documents.

I wanted to get this information out as quickly as possible, so for the first time I've resorted to self-publishing, avoiding the lengthy delays I've experienced with publishing houses in the past. I have to thank Amazon for the ability to do that so easily. If you find this book helpful, I'd really appreciate it if you could write a review and tell your friends. If you think there are omissions or something is unclear, please drop me an email at nick@wifore.com. The advantage of self-publishing is that I can update the book far more readily than with a normal book. I'll also try and answer comments and publish corrections at the book's website at www.bleaudio.com.

All of us involved in the specification development think that Bluetooth LE Audio gives us the tools to develop exciting new audio products and applications. I hope that this book helps to explain those new concepts and inspires you to develop new ideas. If so, the eight years that so many of us have spent working on it will have been time well spent.

The bulk of this book will explain how these new specifications work, how they fit together and what you can do with them, but we'll also take a glimpse at the future in the final chapter. Before jumping into the specifications, it's useful to understand where we are today and what goes into a hearable device, to help understand how everything fits together. That's the purpose of Chapter 1. If you want to get straight to the detail, skip to Chapter 2.

January 2022.

Chapter 1. The background and heritage

Since it was first announced in 1998, Bluetooth® technology has, arguably, grown to be the most successful two-way wireless standard in history. In the wireless standards business, success is normally counted as the number of chips which are sold each year. On that basis, Bluetooth is the winner, with around 4.5 billion chip shipments in 2023. Wi-Fi is close behind, with 4.2 billion, followed by 1.84 billion for all variants of GSM and 3GPP phones and a mere 145 million for DECT.

However, for much of its history, only a small number of those Bluetooth chips were actually used. When Bluetooth technology was first proposed, its developers identified four main use cases. Three of them were audio applications, focussing on simple telephony functions:

- a straightforward wireless headset that was just an extension of your phone, defined in the Headset profile
- an intercom specification for use around the house and in business, and
- a new technology for cordless telephony, hoping to replace the proprietary analogue standards used in the US and the emerging DECT standard within Europe. Its intent was to combine the functions of cordless and cellular in a single phone.

The fourth use case was called dial-up networking or DUN, which provided a means to connect your laptop to your GSM phone, using the phone as a modem to give you internet access wherever you were in the world. As is often the case with new technology standards, none of those four use cases really took off, despite some initial enthusiasm from PC and phone companies. Cordless telephony and intercom failed because they potentially took revenue away from mobile phone operators. Dial up networking worked, but at that point mobile phone tariffs for data were expensive, which encouraged people to use the new Wi-Fi standard instead. Headsets started to sell, but unless you were a taxi driver, you weren't likely to buy one. It became clear that these particular use cases probably weren't the ones that were going to generate scale in the market, so the Bluetooth SIG started work on a host of other features, such as printing and object transfer, none of which attracted much more interest from consumers.

What happened next is what all standards bodies hope for - Government regulations appeared which gave Bluetooth technology a better reason to exist.

At the end of the 1990s, global mobile phone usage exploded as the falling price of both phones and phone contracts changed them from a business tool to a consumer essential. Mobile phone operators started to become High Street names, growing to become substantial, global businesses.

Section 1.1 - The hearing aid contribution

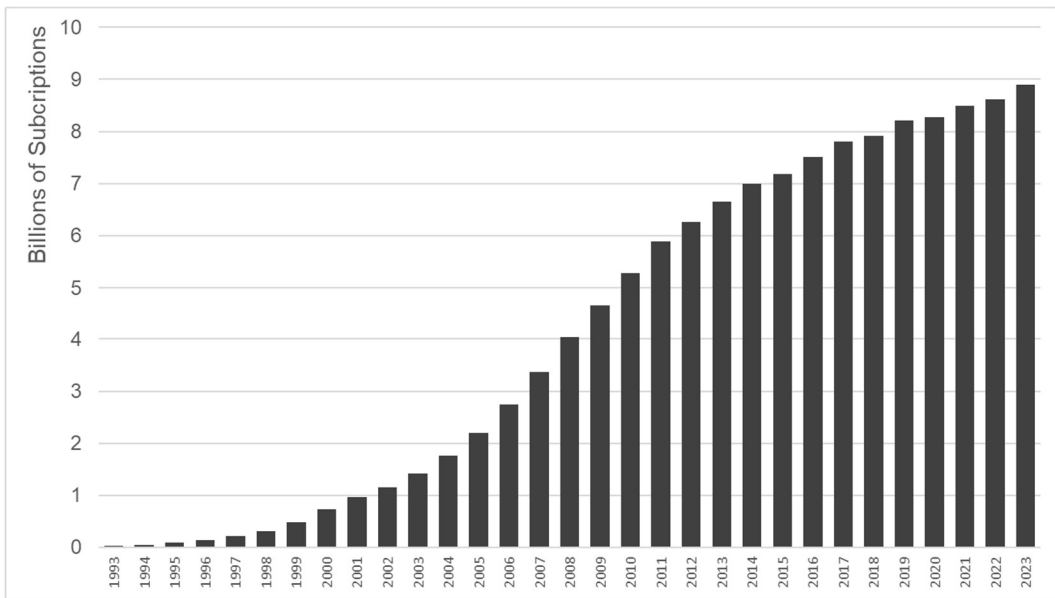


Figure 1.1 Growth of global mobile phone subscriptions

As phone usage increased, so did a concern about where they were used, as more and more road accidents were reported where drivers had been holding their phones and as a result become distracted. Legislators around the world started to propose bans on the use of mobile phones whilst driving. For both the phone industry and the mobile operators this was a potential disaster. In the US, it was reported that almost a third of mobile subscription revenue (which at that time was based on the number and length of phone calls) came from calls made whilst driving. It was a golden egg that the industry could not afford to lose. To save that income, they proposed a compromise to the legislators, which is that safety could be restored if the driver didn't need to hold their phone; instead, the phone call could be taken using a Hands-Free solution, either built into the car itself, or by using a Bluetooth wireless headset.

That was the spur that Bluetooth technology needed. With the new safety legislation coming into effect, phone manufacturers started putting Bluetooth into more and more of their phone models, rather than just the top end ones. The automotive industry began integrating Bluetooth technology into cars and worked with the Bluetooth SIG to develop the Hands-Free profile for that use case. It was a turning point for Bluetooth technology. In 2003, only around 10% of mobile phones were sold which contained a Bluetooth chip – almost all of them in top-end phones. The following year it doubled. The number was to grow every year, as shown in Figure 1.2. By 2008, two thirds of all new mobile phones contained a Bluetooth chip.

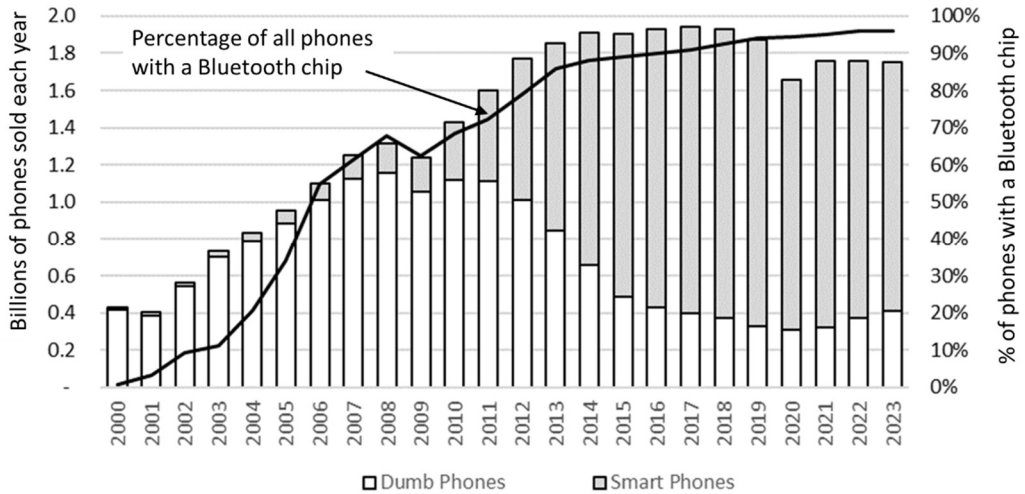


Figure 1.2 Percentage of mobile phones containing a Bluetooth® technology chip

Very few of those chips were actually used for the purposes that were intended. That’s obvious, as only around 14 million headsets were sold in the same year, and that number didn’t grow significantly in the following years. But it was the start of a “free ride” that saw 250 million Bluetooth chips ship in 2005. Those volumes brought competition and manufacturing efficiencies, pushing the price down and starting a virtuous circle where the incremental cost of adding Bluetooth technology was negligible, at least in terms of hardware. The challenge was to find an application which was compelling enough that it encouraged large numbers of people to use it.

Back in 1998, the year Bluetooth technology was announced, the music streaming services we know today, like Spotify and Apple Music, were still ten years away. Ironically, subscription-based music streaming wasn’t an unknown concept – it was over a century old. As far back as 1881, some telephone networks ran a service allowing you to listen remotely to live opera and concerts. But that concept had been trumped by a better organised industry model. The arrival of physical media in the form of wax discs and records, which you could buy and listen to whenever you wanted, killed that original real-time streaming concept. Having discovered that they could own the artists, the recording industry spent the next hundred years doing everything they could to tighten copyright laws around the world to protect their stranglehold on how we listened to music. It was going to prove to be a long dominance. In 1998 we still bought our music in the form of physical recordings. LPs had been almost totally replaced by CDs¹, but around 20% of all the recorded music that we bought that year still came on cassette. Then things changed, thanks largely to the separate efforts of the Fraunhofer Institute for

¹ Ironically, in 2022, LP sales surged to overtake CD sales.

Section 1.1 - The hearing aid contribution

Integrated Circuits (IIS) in Germany and a small Californian startup called Napster.

The Fraunhofer IIS is part of a wider research organisation and a centre of excellence in developing audio codecs – software that compresses audio files so that they are small enough to be wirelessly transmitted or stored as digital files. In 1993, they developed a new audio codec for the international MPEG-1 video compression standard. It was particularly efficient compared to other audio encoding schemes, such as the one used for CDs, and quickly became the standard for audio files transmitted over the internet. It became known as MP3. Without it, we would have had to wait a lot longer for downloadable music.

As a result, the early 2000s became the era of MP3 players and free, downloadable music. Napster burst onto the scene in 1999 and created the first real disruption that the recorded music industry had faced in its hundred year existence. They immediately took Napster to court for copyright infringement, as well as trying to prosecute individual users for downloading music. Consumers voted with their fingers, repeatedly clicking download buttons to get hold of more music. For the first time, music that you could listen to, whenever you wanted, was available for free. By 2000, just a year after it burst onto the market, most of its users probably had more downloaded songs on their PCs than they had physical albums. However, the recording companies turned out to have the better lawyers and Napster lost – the initial attempt to make music free had failed. While the battle to kill off Napster had been dragging through the courts, the next stage of competition had already arrived in the form of Apple's iTunes. It wasn't free, but it was easy to use and legal. Subscribers flocked to the new offering. Six months later, it became even easier, with the launch of the first iPod².

Most of the engineers working on Bluetooth technology were likely to have been Napster and iTunes subscribers, not least to have something to listen to on long-haul flights to standards meetings. By the time the courts had decided Napster's fate, work was already underway to add music streaming to Bluetooth in the form of the Advanced Audio Distribution Profile, better known as A2DP. Despite its far-from-understandable moniker, it was to become the most successful of all Bluetooth profile specifications and cement Bluetooth technology's place as the standard for wireless audio.

A2DP, along with its supporting specifications, was adopted in 2006. Companies like Nokia, who had been heavily involved in its development, expected it to be an instant success, but it proved to be slow in taking off. Consumers didn't see the advantage in buying an expensive wireless set of headphones, with most using the free, corded earbuds which came with every handset, despite their often limited audio quality. Much to the industry's surprise, the initial growth came not from headphones, but from Bluetooth speakers. That new, mobile music market was one where you took your music with you, but didn't necessarily play it while you

² History may well decide that Apple's negotiation of music streaming rights was a more significant industry innovation than the iPod.

were mobile. It quickly became as popular with builders and plumbers as Napster had been with students and Bluetooth engineers. Mobile speakers grew into soundbars as TVs began to include A2DP, but the headphone market remained remarkably resistant to growth until a new service appeared – Spotify.

Spotify (and its US precursor – Pandora) introduced a new business model. You could once again listen to music for free, as long as you accepted advertisements. If you didn't want the interruptions, that was fine – you could get rid of them by paying a subscription. It neatly solved the copyright problem by generating revenue to pay licence fees, regardless of whether users took the subscription or the ad-supported route. It effectively replicated what Napster had done, but found a way to do it legally. It helped that Spotify's launch coincided with the first iPhone, where consumers began to realise that smartphones wouldn't be used predominantly for phone calls. Instead, they'd spend most of their lives doing other stuff, especially as their appearance coincided with the advent of affordable mobile data plans which offered unlimited data usage. Mobile operators were quick to bundle Spotify with their phone contracts and users signed up. Most importantly, it was easy to use. iTunes was still a service where you bought a download and had to make a decision to play it. With Spotify you pressed a button and music appeared. It could be your own playlist; that of a music blogger, a favourite DJ or an algorithm. Its great attraction was that it was frictionless – you pressed a button and music came out of your earbuds until you stopped it. You no longer needed to hold your phone; you could keep in it your pocket or bag, which made it ideal for listening on the move.

At the point that you don't need to hold your phone, an earbud cable becomes a nuisance. It was the nudge users needed to persuade them to start buying Bluetooth headphones. Branded Bluetooth headphones sales started to rise, as manufacturers saw customers cut the cable. By 2016, Bluetooth headphones were outselling wired ones.

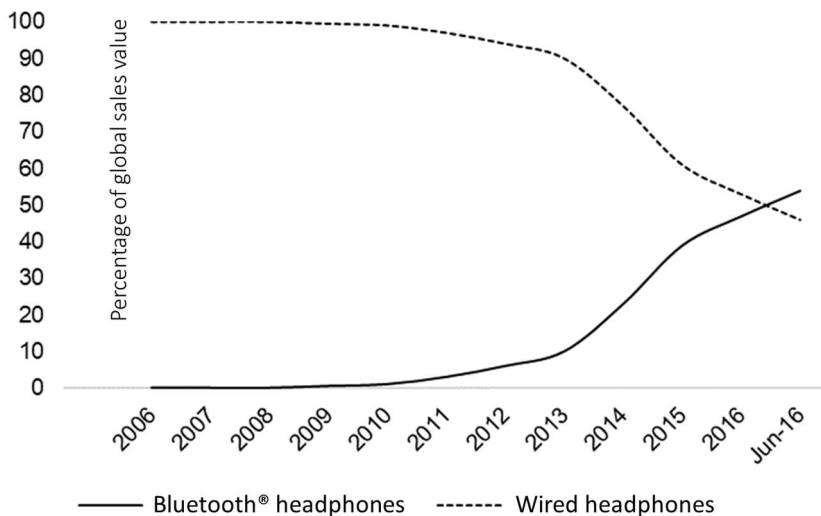


Figure 1.3 The transition from wired to Bluetooth® headphones

Section 1.1 - The hearing aid contribution

Figure 1.4 shows how much the Spotify experience helped drive that change. Over the ten years since 2006, when Spotify and the A2DP specification independently appeared on the market, the growth in Bluetooth headphones has almost exactly tracked that of Spotify subscribers.

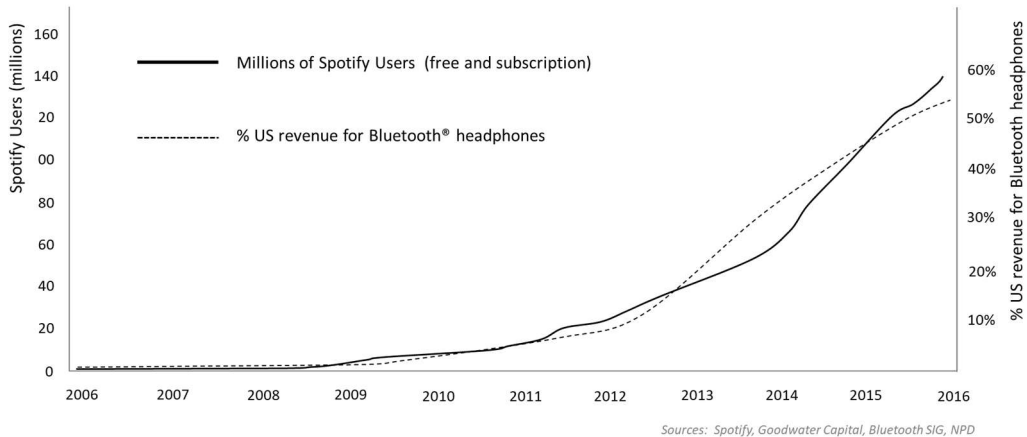


Figure 1.4 The growth of Spotify subscriptions and Bluetooth® headphones

The growth in chip sales for wireless headphones drove innovation. By 2010, Bluetooth silicon companies were shipping over 1.5 billion chips per year and were looking for more ways to differentiate their products. The new Bluetooth LE standard had just launched, but it wouldn't be until the end of 2011 that it appeared in the iPhone 4s, and several more years before a new generation of wristbands started to use it. In the interim, Cambridge Silicon Radio (CSR, who were subsequently acquired by Qualcomm), had become the leading player in chips for Bluetooth audio devices, and were looking at how they might extend the functionality of the Bluetooth audio profiles.

Why would they want to do that? The problem they saw was that both of the two main Bluetooth audio standards were written for very specific use cases, which hadn't anticipated the future. The result of that is that they behave very differently. HFP concentrates on low latency, bidirectional, mono voice transmission, whereas A2DP supports high quality music streaming to a single device with no return audio path. Neither of those profiles were easily extensible, which limited what you could do with them. CSR were trying to push the boundaries of wireless audio. They had already been successful in developing a more efficient codec than the SBC codec mandated by the Bluetooth specifications, adding their own enhanced AptX codec into their chips. Now they decided to push further ahead and see if they could discover a way to allow the A2DP specification to be used to stream stereo to two independent earbuds. That had an important commercial advantage, as earbud manufacturers would need to purchase twice as many Bluetooth chips – one for each ear.

Their efforts succeeded and started a new era for Bluetooth audio, which would lead to

massive growth. The consumer uptake of wireless earbuds meant that Bluetooth technology decoupled itself from Spotify's growth and gained a new momentum of its own. CSR's innovation was to develop a way for a single earbud to receive an A2DP stream and forward one channel of the stereo stream, together with timing information, to a second earbud. Using the timing information, the first earbud could delay rendering its audio channel until it knew the second earbud had received its audio data and was ready to render its stream. As far as the user was concerned, it appeared as if each earbud was receiving its own left or right channel. It was not quite as simple as that to make it work, as we'll see later, but it would become a game changer.

The first movers to exploit this new innovation were not the big companies. At the start of 2014, two companies in Europe – Earin in Sweden and Bragi in Germany kicked it all off with crowdfunding campaigns for stereo wireless earbuds. Earin's offering was a pair of wireless earbuds that used the new chipsets to push the boundaries of what could be done. However, it was Bragi's Dash that really caught the imagination. In the spring of 2014, they broke the Kickstarter record for the most successful crowd-funding campaign up to that date, raising over \$3.3 million for their Dash wireless earbuds. It was an amazing piece of engineering, which promised to cram almost every feature you could think of into a pair of earbuds. It raised the bar of what you can do with something in your ear, opening up a whole new market sector for which I coined the name "hearables". It was a massively ambitious concept, and to their credit, they managed to ship the Dash. Despite an enthusiastic following, it showed that it's not enough to just integrate the technology – you need to find a use for it. Despite providing an SDK for software developers, the Dash failed to gain enough traction with consumers. In the following two years, other crowdfunded projects dreamt up even more complexity and managed to attract around \$50 million dollars of funding. Many failed to deliver, coming to realise just how hard it is to cram that amount of technology into a small earbud. Hardware is hard. Most of these startups fell by the wayside, and even Bragi was forced to make the difficult decision to move out of hardware, selling its product range and concentrating on embedded operating systems for other mainstream audio products, where it remains a major force in hearable design.

Gradually, bigger names started to dip their toes into the hearables pond, utilizing their deeper resources to make these difficult products work. Then, in September 2016, Apple launched its AirPods. Although initially derided by many journalists, consumers loved them. In just two years they became the fastest selling consumer product ever, and have sold almost 600 million pairs since they were launched. Other brands rapidly followed on, with China's chip vendors jumping on the bandwagon. Back in 2014, when Bragi and Earin were setting out the future of the market, there were only four or five silicon vendors making Bluetooth audio chipsets. Today there are over thirty. Covid lockdowns accelerated sales, which have remained strong. It is estimated that around 500 million earbuds were shipped in 2023, with a prediction that the number will continue to grow over the next ten years. Those figures are for earbuds based on the classic HFP and A2DP profiles. As new Bluetooth LE Audio products start to ship, with their additional functionality, broadcast features and enhanced

Section 1.1 - The hearing aid contribution

battery life, the numbers are likely to exceed those predictions.

However, wireless audio is not just about earbuds. Most of the growth in Bluetooth audio has been driven by music streaming, which in turn has been driven by the ready availability of content from services like Spotify, Apple Music and Amazon Prime music. The arrival of streaming video has been equally popular, with wireless earbuds becoming the device of choice for listening. Consumers like ease of use, and in most cases that translates into consuming content, not generating it themselves, although applications like TikTok are showing that if content generation can be made simple and amusing, users will produce and share their own.

In this evolution, voice, mainly the preserve of voice calls, had looked as if it was becoming the poor relation. That changed when Amazon launched Alexa. Despite some reservations about privacy, consumers started talking to the internet. Since then, voice recognition has seen a renaissance, to the point where most home appliances now want to talk to us, and for us to talk to them. Those applications are likely to grow with the introduction of the new features supported by Bluetooth LE Audio, which are covered in this book. With broadcast topologies and the ability to prioritise which devices can talk to you, it becomes possible to add extra functionality and new opportunities to voice to machine communication. It may be the saviour of the Smart Home industry.

The speed of development surrounding the introduction of earbuds, and Apple's AirPods in particular, has been amazing. We have seen many new companies developing Bluetooth audio chips, advances in miniature audio transducers and MEMS³ microphones, along with a massive growth in the number of companies providing advanced audio algorithms to enable features like active noise cancellation, echo cancellation, spatial sound and frequency balancing.

Although the Hands-Free Profile and A2DP continue to serve us well, both were designed for a simple peer-to-peer topology. Referred to as Bluetooth Classic Audio, their design assumes a single connection between two devices, where each individual audio data packet is acknowledged. That doesn't work if there are two separate devices that want to receive the audio stream. Because of that, today's stereo earbuds depend on proprietary solutions which generally involve the addition of a second radio to communicate between the left and right earbuds to ensure that both of them play the audio at the right time. Another limitation is that the Hands-Free Profile was not designed for the wide range of cellular and Voice over IP telephony applications we use today. Having just these two, independent, dedicated audio profiles has led to multi-profile problems when users want to swap from one application to another. That's before you add in new requirements which have emerged for concurrent voice

³ MEMS stands for Microelectromechanical Systems, which in this case refers to microphones which have been made by etching physical structures into a silicon wafer. It's a very efficient way of making small, highly accurate sensors.

control and shared audio.

We're all using wireless audio more frequently throughout our daily lives, whether that's to talk to each other, or insulate ourselves from the outside world. To support this change in behaviour we need to think less about connections between individual devices that are normally used together, like a headset and the phone. Instead, we need to be far more aware of a wider audio ecosystem, where we have different devices that we might wear on our ears or have around us during the course of a day, constantly listening to and changing what they do with other devices. For that to work seamlessly, control needs to become far more flexible. This is the background that led to the development of Bluetooth LE Audio.

The Bluetooth SIG realised that their audio specifications needed to evolve and adapt, both to cope with current requirements and also to look to the increasingly diverse range of things we're doing with audio, as well as what we can expect to appear over the next 20 years. There are some very similar requirements that come up in many of the use cases. Designers want lower power. Not just for extended battery life, but to be able to support more processing for noise reduction and the other interesting audio algorithms that are emerging, such as detecting oncoming traffic or relevant conversation. For other applications, they want to reduce latency, particularly for gaming or listening to live conversations or broadcasts. There's also the never-ending quest for higher audio quality. The Bluetooth LE Audio working groups developing the specifications have had the task of coming up with new standards that support these requirements, as well as all of the topologies that are envisioned, without having to rely on non-interoperable extensions. The aim is to allow the industry to move on from the position it's in today, which relies on proprietary implementations, to one where you can mix and match devices from different manufacturers.

1.1 The hearing aid contribution

It surprises many people to hear that a lot of this innovation was kick-started by the hearing aid industry. Hearing aids have needed to solve the issues of audio quality, latency, battery life and broadcast transmissions for many, many years. They are worn for an average of nine hours a day, so battery life is critical. During that time hearing aids are constantly amplifying and processing ambient sound so that the wearer can hear what is happening and being said around them. They typically include multiple microphones to allow audio processing algorithms to recognise and react to the local audio environment in order to filter out distracting sound. In public spaces, where the facility is available, they can connect to a system called telecoil, essentially induction loops, which are used in theatres, public transport and other public areas to transmit audio to assist listeners with hearing difficulties and to provide information. These are broadcast systems which can cope with hundreds of people within the transmission area of the telecoil, or allow private conversations using very small loops.

Hearing aid users have always wanted to be able to connect to phones and other Bluetooth devices, but the power consumption of traditional HFP and A2DP solutions was a challenge. In 2013, Apple launched a proprietary solution based on the Bluetooth LE specification,

Section 1.2 - Limitations and proprietary extensions

adding an audio stream which could connect to special Bluetooth LE chips in hearing aids. It was licensed to hearing aid manufacturers, and appreciated by consumers, but it only worked with iPhones and was unidirectional.

Although welcoming the development, the hearing aid industry was concerned that an Apple-specific solution was not inclusive. They wanted a global standard which would work with any phone or TV, and which could also replace the ageing telecoil specification, which dated back to the 1950s. In 2013, representatives of all of the major hearing aid companies sat down with Bluetooth SIG's Board, and came up with a joint agreement to provide resources to help develop a new low power Bluetooth standard for audio to bring interoperability to the hearing aid ecosystem. Fairly soon after the development work began, many consumer audio companies started looking at the hearing aid use cases and realised that they were equally applicable to the consumer market. Although the audio quality requirements for hearing aids were less stringent (as their users have hearing loss), the use cases, which combined ambient audio, Bluetooth audio and broadcast infrastructure, were far more advanced than the ones currently covered by HFP and A2DP. They had the potential to solve many of the known problems and limitations with the current audio specifications.

As more and more companies got involved, the project expanded. Over the eight years that the work has taken, the Bluetooth LE Audio initiative has evolved into the largest specification development project that the Bluetooth SIG has ever done. The resulting specifications cover every layer of the Bluetooth standard, and consist of over 1,250 pages of text in new and updated documents. Additional specifications for new applications are already well advanced.

1.2 Limitations and proprietary extensions

1.2.1 Apple's Made for iPhone (Mfi) for hearing devices and ASHA

In 2013, Apple launched its own proprietary Bluetooth LE solution for hearing aids, which it licensed to hearing aid manufacturers. Developed in conjunction with one of its silicon partners, it added extensions to the Bluetooth LE protocols to allow unidirectional transmission of data between a phone and one or two hearing aids. An app on the phone allowed the user to select which hearing aid to connect to, as well as allowing them to set volume (either independently, or as a pair) and to select a variety of presets, which apply pre-configured settings on the hearing aids to cope with different acoustic environments. The Mfi hearing devices solution worked on iPhone 5 phones and iPad (4th generation) devices and subsequent products.

One of the popular features that Mfi supports is "Live Listen", which allows the iPhone or iPad to be used as a remote microphone. For hearing aid wearers, this lets them place their phone on a table to pick up and stream a conversation. Remote microphones are useful accessories for hearing aid users and the Live Listen feature provides this without the need to buy an additional device.

Early in 2021, Apple announced that their Mfi for hearing devices would be upgraded to allow bidirectional audio, bringing Hands-Free capability. This year, they also announced that their latest AirPods could be upgraded to act as hearing aids for users with low levels of hearing loss, and that the iOS18 update would include a hearing test.

Apple's motives weren't entirely altruistic. Accessibility regulations in many countries forced them to include a telecoil in their phones, which adds cost and constrains the physical design. They hoped that regulators would accept a Bluetooth solution as an alternative. Regulators disagreed, so their handsets still include the telecoil. Nokia had a similar desire and was active in supporting the development of the Bluetooth LE Audio specification before they withdrew from making handsets.

Apple's Mfi for hearing solution only works with iPhone and iPads, leaving Android owners with no solution. In parallel with the Bluetooth LE Audio development, Google and hearing aid manufacturer GN Resound worked together to develop an open Bluetooth LE specification called ASHA (Audio Streaming for Hearing Aids) which is a software solution which works on Android 10 and above. It provides a different proprietary extension to Bluetooth LE to support unidirectional streaming from any compliant Android device to an ASHA hearing aid.

ASHA has provided a welcome fill-in for Android users in the gap before Bluetooth LE Audio starts to appear in phones. It's likely that hearing aid manufacturers who currently support Mfi for hearing aids or ASHA will continue to do so. They will extend their support by adding Bluetooth LE Audio to provide the widest choice for consumers. At the end of the day, it is just another protocol, which means more firmware and memory. However, it is likely that most new development will move towards the globally interoperable Bluetooth LE Audio standard.

1.2.2 True Wireless Stereo

A practical solution to transmit a stereo stream to two earbuds was developed by Cambridge Silicon Radio around 2013. After they were acquired by Qualcomm it was rebranded as TrueWireless, which is the phrase that most of the world now uses for stereo earbuds. As competitors looked at how they could emulate the success of Apple's AirPods, which use Apple's own proprietary chipset, Qualcomm responded by providing a viable alternative for every other company that wanted to develop a competing earbud. The name True Wireless Stereo or TWS quickly became established and applied to almost every new product, regardless of whose chip was in it.

The main obstacle to using A2DP with separate earbuds is that it was designed for single point-to-point communications. The Bluetooth SIG did provide guidance for how streams could be sent to multiple Bluetooth LE Audio sinks in a white paper titled "White paper on usage of multiple headphones" back in 2008, but it avoided the question of synchronization, assuming that each audio sink could make up its own mind about when to render the stream.

Section 1.2 - Limitations and proprietary extensions

For earbuds, as soon as the left and right streams move out of sync, you get a very unpleasant sensation of sound moving around inside your head. The white paper approach also relied on modifications to the A2DP specification at the audio source. That's a problem, as it means that earbuds or speakers that didn't adopt them wouldn't be compatible. To be successful in the market there needed to be a solution which would work with any audio source, which effectively meant that the solutions needed to be implemented solely in the audio sinks – the earbuds.

The solution that CSR originally came up with is known as the replay or forwarding approach, as shown in Figure 1.5.

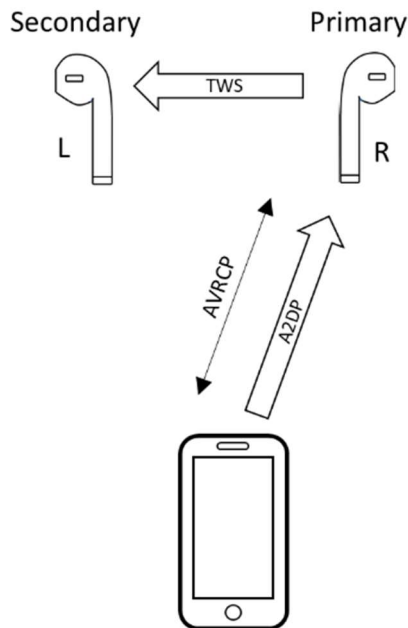


Figure 1.5 Replay scheme for TWS

Before they connect to the phone, the two earbuds pair with each other (which may be done at manufacture). One is set to be the primary device, the other as a secondary. To receive an A2DP stream, the primary device pairs with the audio source, appearing as a single device receiving a stereo stream. When audio packets arrive, it decodes the left and right channels and, in the example shown above, relays the left channel directly to the secondary earbud. This may be possible using Bluetooth technology, but if the earbuds have small antennas, this may be problematic, as the head absorbs the 2.4GHz signal very well. Many companies discovered this when they first tested their devices outside. Indoors, reflections from walls and ceilings will generally ensure that the Bluetooth signal gets through. Outside, with no reflecting surfaces to help, they may not. To compensate for this, a second radio is typically added to TWS earbuds to get around the absorption problem that arises from having a brain between your ears. The most popular option is Near Field Magnetic Induction (NFMI), which is an efficient low power solution for short range audio transfer. Other chip vendors have

used similar Low Band Retransmission (LBRT) schemes, which operate at a sub-GHz frequency. It's a problem that hearing aid developers solved over a decade before, but which was new for the consumer earbud industry.

As the primary earbud is in charge of the timing for the audio relay, it knows exactly when the secondary earbud will render it. It needs to delay the rendering of its audio stream to match. That is one of the issues with the relay approach, as the relay process and buffering adds to the latency of the audio connection. For most applications it is unlikely to be noticed by the user, not least because A2DP latency will normally dominate.

The relay approach also works for HFP, although in many cases, only the primary device is used for the voice return path.

Volume and content control, such as answering a call or pausing music still uses the Audio/Video Remote Control Profile (AVRCP) over the connection to the primary earbud. The second radio, when present, can also be used to relay user interface and AVRCP controls, such as pause, play and volume, between the primary and secondary device.

More recently, companies have started moving to a sniffing approach, illustrated in Figure 1.6

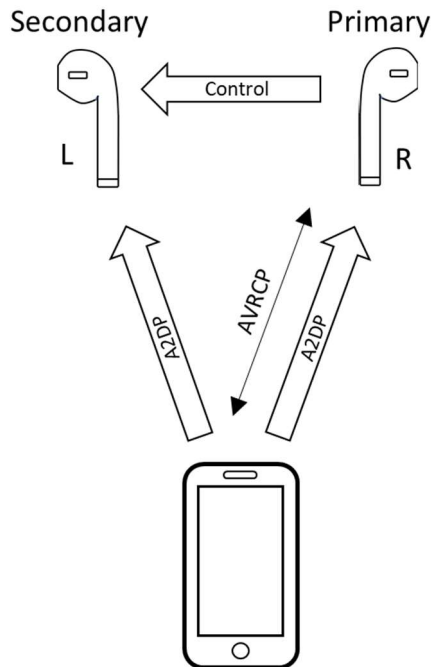


Figure 1.6 The sniffing approach for TWS

Here, the added latency of the relay approach is removed by having both earbuds listen to the primary A2DP stream. Only one of the earbuds (the primary device) acknowledges receipt of the audio data to the phone. Normally, the secondary device wouldn't know where to find the audio stream, or be able to decode the audio data. In this case, the primary device provides

Section 1.2 - Limitations and proprietary extensions

it with that information, along with the necessary synchronization timing, either through a Bluetooth link or a proprietary sub-GHz link. That link is configured by the manufacturer, so there's no chance of other devices being able to pick up the A2DP stream.

As both earbuds receive the same A2DP stream directly, this is potentially a far more robust scheme, particularly if implemented as a Bluetooth only solution. Where there is no relaying of the audio stream, the latency is appreciably better.

Both of these schemes require clever extensions at a fairly low level of the Bluetooth stacks in the earbuds, so have largely been the domain of chip vendors. Apple's success has spurred competing silicon providers to innovate, with the result that around a dozen different True Wireless schemes are now in existence, built on variants of these two techniques, with some including additional features like primary swapping, so that the two earbuds can change roles if one loses the link.

The problem with these proprietary approaches is that they can fragment the market. Apple, Qualcomm and others have all filed patents for their solutions, which results in their competitors spending time and effort looking at how to get around these patents rather than innovating to drive the market forward. It also means that it is almost impossible to mix earbuds from two different manufacturers, as they're likely to use different TWS schemes. That may not be an issue for consumer TWS earbuds, where they are always bought as a paired set, but it is for hearing aids, where different types may be needed for left and right ears. It makes it difficult to extend the scheme to speakers, where surround sound systems often combine speakers from different manufacturers. Although proprietary solutions can spur market growth in its early stages, they rarely help its long term development. Which is where Bluetooth LE Audio comes in.

1.2.3 Shared listening and Auracast

One of the major use cases for Bluetooth LE Audio is to enable Bluetooth LE Audio to be shared. That may be with a friend, when you're listening to a song on their phone; with your family, when you're watching the TV, or within a much larger group, such as in a sports bar. It's something that hearing aid users already have access to, with telecoil audio loop systems, but the goal for Bluetooth LE Audio was to make it available for all as a standard feature in phones, TVs and public spaces, providing a scalable solution to transition from one to many listeners. The scalability doesn't just apply to the number of listeners, but also the number of transmitters. Telecoil's audio loops can't overlap, so where they are fitted, they only carry one audio stream. In contrast, multiple Bluetooth Broadcast Transmitters can work alongside each other in the same space. That might be multiple TVs in a bar or a gym; it could be different background music in a restaurant or coffee shop, or different language tracks for a film in a cinema. Equally, multiple people in a train or an airport lounge could be sharing their individual audio streams with family or friends – with all of those streams private to their group. As the expectation is that all Bluetooth chips will include this new broadcast functionality, it brings a new experience at no additional cost.

The low cost implication of broadcasting audio is likely to lead to some major changes in the way that audio is used, not just for individual users, but also for audio infrastructure. Simple Broadcast Transmitters should be no more expensive than today's Bluetooth speakers, allowing any venue to add a broadcast capability for a few tens of dollars. It also makes it very cost-effective to develop wireless speakers for public venues which remove the need for wiring. A bar, restaurant, school or community hall can simply add speakers wherever they want them, as long as a power socket is available – there's no more need for cabling. That's a market which is even bigger than the smartphone market.

History has shown that technologies which grow to billions of units need interoperability. Unless devices made by multiple different manufacturers work together, they don't achieve critical mass. For new specifications and applications, that can take years to develop. To try to reduce that timescale, the Bluetooth SIG has developed and published the Auracast™ Simple Transmitter Best Practices Guide for manufacturers and developers of broadcast audio applications, under the trade name of Auracast™. Products which conform to these guidelines can carry the Auracast™ logo, which tells consumers that the products, and the venue in which they are installed will work with their hearing aids and earbuds. We'll look at the details of these in more detail in Chapters 12 and 14.

1.3 What's in a hearable?

Most of this book is about the fine detail of the Bluetooth LE Audio specifications, which companies will use to bring new experiences to audio products. Before diving into that, it's useful to look inside a typical earbud to see what's in it, as we'll be referring back to the different hardware elements as we progress through the specifications.

The relatively recent arrival of wireless earbuds was limited by the availability of chips which could provide a way to support separate left and right stereo streams. But that's not the only reason. As all of the original startup companies who entered this market discovered, packing all of the functionality that you need to reproduce decent audio into something as small as an earbud is hard. In fact, it's very hard. Adding Bluetooth technology to that, along with any additional sensors to support it, becomes incredibly hard. Only around 25% of the crowdfunded hearable companies ever managed to ship a product. Even companies like Apple had to commit almost five years of development to bring about the AirPods. They even resorted to designing their own Bluetooth chips to get the performance which made AirPods such a success. That's something that only the largest earbud companies – Apple and Huawei, have had the resources to do.

It's a story that hearing aid manufacturers know well, as they've been perfecting the miniaturization of hearing aids for many years. They work with customised chips to optimise performance, resulting in devices which run for days on small, primary zinc-air batteries. There are a surprising number of elements in a Bluetooth hearing aid, as shown in Figure 1.7, which represents a fairly basic design. It's a very similar architecture to an earbud.

Section 1.3 - What's in a hearable?

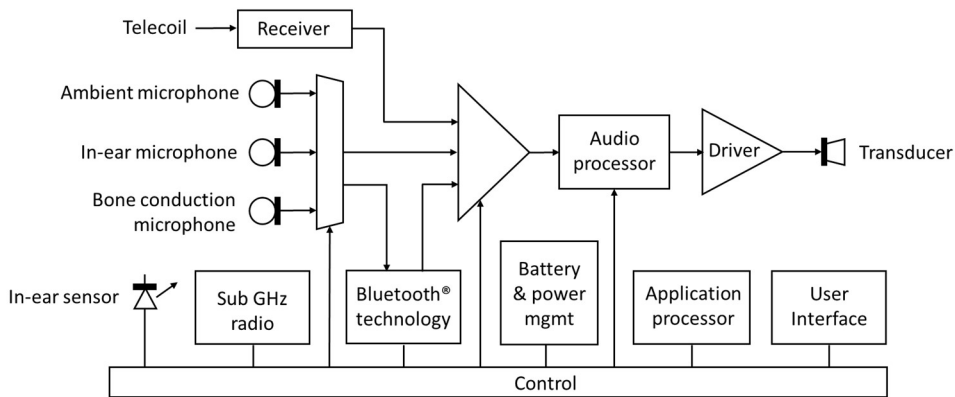


Figure 1.7 Architecture of a simple Bluetooth® hearing aid

When you take a hearing aid or earbud to pieces, the first surprise most people encounter is the number of microphones in them. To perform active noise cancellation, you need one microphone monitoring the ambient sound and a second one in the ear canal. If you want to pick up the user's voice to send over the Bluetooth link to their phone, there will normally be a bone conduction microphone helping to pick up and isolate the spoken voice from the ambient. That's the minimum. However, it's common to include additional microphones to generate a beam-forming array to improve directionality. Other audio algorithms may be included to detect the type of environment and further enhance the user's voice. The latter is important, as the microphones in earbuds and hearing aids aren't located as close to the mouth as designers would want them to be – they may often be behind the ear.

We're about to see new regulations come into effect to measure the sound level in the ear canal and warn users about potential hearing damage, which will probably lead to even more internal microphones. The good news is that there has been a lot of development in MEMS microphones in the last few years, partly driven by the demands of voice assistants. These innovations are enhancing directionality and beam steering, so that they can follow you around the room. The advantage of MEMS over traditional microphone structures is that you can integrate digital signal processors into the microphone itself, which reduces size and power consumption. It's not unusual to find four or more microphones in the latest hearables. These inputs need to be mixed and dispatched to the different functions of the hearing aid.

If the device contains both Bluetooth technology and telecoil receivers, the appropriate audio needs to be routed from them. For earbuds which send control signals or audio streams to a second earbud, these all need to be routed via a sub-GHz radio (normally NFMI), whilst the primary signal is buffered to synchronize the rendering time at both earbuds.

At the output, audio transducers have become smaller, with traditional open coil structures being challenged by micro-miniature balanced armature transducers and audio valves. The market is also seeing the appearance of MEMS speakers, offering higher sound levels. Whilst

they are probably more suited to headphones at this stage, the technology is likely to migrate to earbuds.

Looking at the new use cases that are being made possible with Bluetooth LE Audio, this processing can become very complex. A noise cancelling earbud receiving a Bluetooth stream and allowing voice commands to be transmitted at the same time will need to separate the voice component from the ambient sound (and apply echo cancellation) before transmitting it, whilst at the same time suppressing the ambient sound that is mixed with the incoming Bluetooth signal. If the incoming Bluetooth stream is being broadcast from the same source as the ambient, such as when you're in a theatre, conference room, or watching TV, then this all needs to be done whilst maintaining an overall latency of around 30 msecs. That is challenging.

Managing all of this needs an application processor, which controls the specialised audio processing blocks. To minimise power and latency in hearing aids, these are normally implemented in hardware rather than in a general purpose Digital Signal Processor (DSP). The application processor will also normally control the user interface, where commands may come from buttons or capacitive sensors on the hearing aid, via the Bluetooth interface, or from a remote control, which may be using either a Bluetooth technology or proprietary sub-GHz radio link. Most devices include an optical sensor to detect when it is removed from the ear, so that it can be placed into a sleep state. Finally, there is a battery and power management function. Many hearing aids still use zinc-air batteries, which provide a better power density than rechargeable batteries. That provides two main advantages – the battery life is longer, and they weigh less. The weight is important if you're wearing a hearing aid all day, which is why most modern hearing aids weigh less than 2 grams – around half the weight of an AirPods.

That's just the electronics. Fitting all of this into an earbud or hearing aid is a further challenge, pushing the limits of flexible circuitry and multi-layer packaging. Designers need to make it comfortable, ensure it doesn't fall out of your ear, but also accomplish all of this whilst maintaining a good auditory path, so that neither the fit, nor the electronics packed into it, affects the quality of the sound. You also need to consider the question of whether the hearing aid occludes, i.e., whether it blocks your ear to stop ambient sound, or is open to allow you to hear both. Until recently it was felt that occlusion was necessary if you wanted to have effective ambient noise cancellation, but a few recent innovations suggest that may no longer be the case. Completely blocking the ear canal can cause issues with a build-up of humidity, especially for prolonged wearing, so we will probably see more designs taking the open direction.

None of this is easy, which is one of the reasons that hearing aids remain expensive. However, a growing number of chip companies are offering reference designs which have already done much of the work. Along with partner programs with specialist consultancies offering design expertise to reduce the time to market, this is resulting in the consumer earbud market growing at a phenomenal pace. But we're still just at the beginning of the possibilities for what you

Section 1.3 - What's in a hearable?

can put in your ear.

The most ambitious hearable which has shipped so far was Bragi's Dash. As well as providing true wireless stereo, they decided to add an internal MP3 player and flash storage, allowing you to leave your phone at home while you're out running or in the gym, yet still be able to listen to your stored music directly from the earbuds.

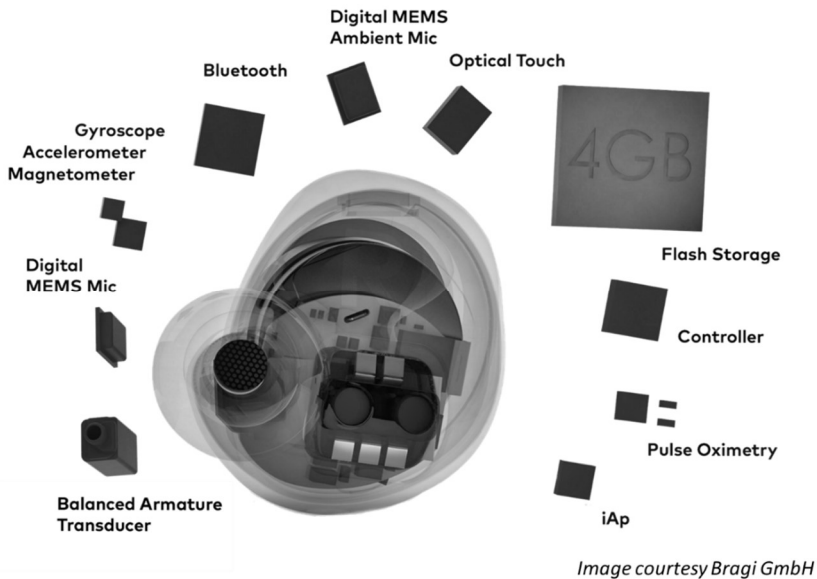


Figure 1.8 The Dash earbud from Bragi - the first real hearable

Recognising that the ear is the best place on the body for most physiological sensors, Bragi equipped the Dash with a plethora of sensors and features: a total of nine degrees of freedom movement sensing with an accelerometer, magnetometer and gyroscope, a thermometer, a heart rate monitor and a pulse oximeter. They produced a very nice graphic showing all of the elements with their relative sizes, which is represented in Figure 1.8. It was a stunning achievement, but sadly, it was more than the market wanted at that time. As fitness wristband manufacturers discovered, it's surprisingly difficult to keep customers engaged with their health data. Unlike music, where they just devour someone else's content, health data needs a lot of analytics to turn it into a compelling story for the user.

That's a Catch 22 which is illustrated in Figure 1.9. You need to capture a lot of data before you have enough to turn into anything valuable. During that time, you need to employ some very expensive data scientists to try and develop some compelling feedback, pay for the ongoing cost of cloud storage and analytics, as well as app updates for every new version of the phone operating systems you support. There is no guarantee that this will ever provide feedback which is compelling enough for the user to pay a monthly subscription to support the ongoing development costs. But, without that insight, users will give up using the product, which is why so many fitness bands are now sitting at the back of bedroom drawers. It doesn't

help that very few of the companies making these devices have a data analytics business background, rather than a pure hardware business model. In general, their background is selling you a product, with the expectation that they make a profit from the sale, with no further costs other than a small proportion of warranty returns. In contrast the service model expects a large percentage of users to pay a regular fee for ongoing feedback. Very few companies have managed to make that work, other than in very niche applications. It is still something that many desire, but has proven difficult to achieve.

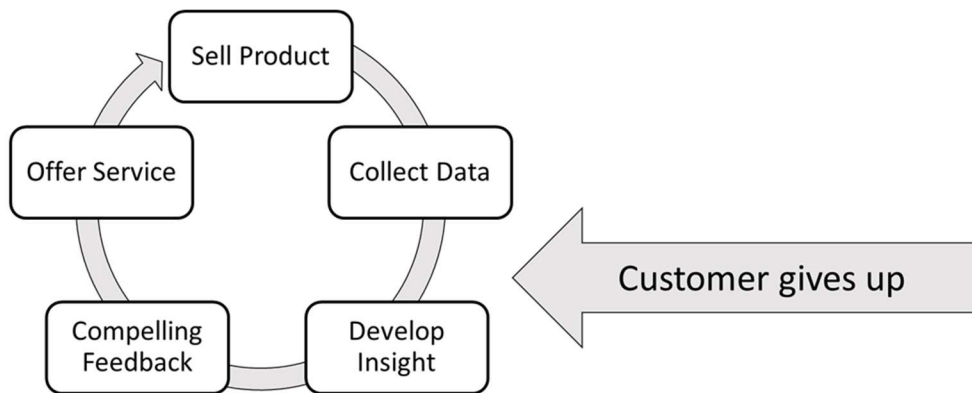


Figure 1.9 The Catch 22 of health and fitness data

The difficulty of developing an insight business model is why almost all of the billion plus earbuds which have shipped so far have concentrated on just one thing – playing content, where the user has a choice of multiple, mature services to choose from. I suspect that in time we will see sensors return to hearables, as the ear is the best place for a wearable device to measure biometrics. It's stable, it doesn't move much, it's close to blood flow and provides a good site to measure core temperature. It's everything that you want in terms of a location for a wearable health sensor, unlike the wrist, which is just the opposite, lacking these attributes. However, the wrist is the site targeted by most of today's health sensors, as it's easier. Putting additional sensors into an earbud remains a major challenge, purely because of space.

Many of the early entrants have learnt the lesson that data is hard, which means that health sensors are often added as minor features, allowing companies with the analytic resources time to develop that compelling feedback. It's what we've seen Apple do with the Watch. It's a long slow business, which will only pivot to a health-led one when we start to see real health benefits emerge. Fortunately for earbuds, there is already a compelling reason for consumers to buy them, which means that hearables can be a useful platform to experiment with other things, as long as users consent.

All of this translates to a vast amount of excitement in the market. Earbuds are the fastest growing consumer product ever and the pace shows no sign of slowing. For the rest of this book, we'll look at how Bluetooth LE Audio can add to the excitement and increase that rate of growth.

Chapter 2. The Bluetooth® LE Audio architecture

Bluetooth specification development follows a well-defined process. It starts off with a New Work Proposal, which develops use cases and assesses the market need for any new feature. The New Work Proposal is usually generated by a small study group consisting of a few companies who want the feature and is then shared and appraised by any others who are interested in it. At that point, other Bluetooth SIG members are asked if they're interested in helping develop and prototype it, in order to see if there's enough critical mass for it to happen.

Once that level of commitment has been demonstrated, the Bluetooth SIG Board of Directors reviews it and assigns it to a group to convert the initial proposal into a set of requirements and to put more flesh on the use cases. Those requirements are reviewed to make sure they fit into the current architecture of Bluetooth technology without breaking it, and then development begins. Once the specification is deemed to be more or less complete, implementation teams from multiple member companies develop prototypes which are tested against each other in Interoperability Test Events, which check that the features work and meet the original requirements. This also provides a good check that the specifications are understandable and unambiguous. Any remaining problems get addressed at that stage, and once that's done and everything is shown to work, the specifications are adopted and published. Only at that stage are companies allowed to make products, qualify them and start selling them.

Although we always try to avoid specification creep during the process, we almost always fail, so new features tend to get added to the original ones. That's been particularly true in the evolution of Bluetooth LE Audio, as it's evolved from being a moderately simple solution for hearing aids, into its current form, which provides the toolkit for the next twenty years of Bluetooth audio products. To help understand why we have ended up with over twenty new specifications, it's useful to look at that journey from the original use cases to see how the final architecture was determined

2.1 The use cases

In the initial years of Bluetooth LE Audio development, we saw four main waves of use cases and requirements drive its evolution. It started off with a set of use cases which came from the hearing aid industry. These were focused on topology, power consumption and latency. As more companies joined in, the scope broadened, going through the following phases of requirements:

- Hearing aid requirements
- Doing everything that HFP and A2DP can do
- Evolving beyond HFP and A2DP
- Addressing future audio applications

Section 2.1 - The use cases

2.1.1 Hearing aid requirements - the use cases

The topologies for hearing aids were a major step forward from what the Bluetooth Classic Audio profiles do, so we'll start with them.

2.1.1.1 Basic telephony

Figure 2.1. shows the two telephony use cases for hearing aids, allowing hearing aids to connect to phones. It's an important requirement, as holding a phone next to a hearing aid in your ear often causes interference.



Figure 2.1 Basic Hearing Aid topologies

The simplest topology, on the left, is an audio stream from a phone to a hearing aid, which allows a return stream, aimed primarily at telephony. It can be configured to use the microphone on the hearing aid for capturing return speech, or the user can speak into their phone. That's no different from what Hands-Free Profile (HFP) does. But from the beginning, the hearing aid requirements had the concept that the two directions of the audio stream were independent and could be configured by the application. In other words, the stream from the phone to the hearing aid, and the return stream from hearing aid to phone would be configured and controlled separately, so that either could be turned on or off. The topology on the right of Figure 2.1 moves beyond anything that A2DP or HFP can do. Here the phone sends a separate left and right audio stream to left and right hearing aids and then adds the complexity of optional return streams from each of the hearing aid microphones. That introduces a second step beyond anything that Bluetooth Classic Audio profiles can manage, requiring separate synchronized streams to two independent audio devices.

2.1.1.2 Low latency audio from a TV

An interesting extension of the requirement arises from the fact that hearing aids may continue to receive ambient sound as well as the Bluetooth audio stream. Many hearing aids do not occlude the ear (occlude is the industry term for blocking the ear, like an earplug), which means that the wearer always hears a mix of ambient and amplified sound. As the processing delay for ambient sound within a hearing aid is minimal – less than a few milliseconds, this doesn't present a problem. However, it becomes a problem in a situation like that of Figure 2.2, where some of the family are listening to the sound through the TV's speakers whilst the hearing aid user hears a mix of the ambient sound from the TV as well the same audio stream through their Bluetooth connection.

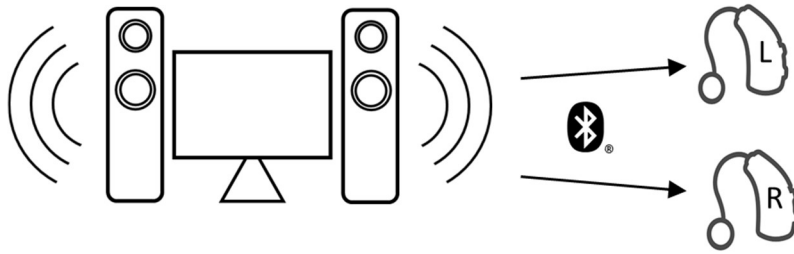


Figure 2.2 Bluetooth® LE Audio streaming with ambient sound

If the delay between the two audio signals is much more than 30 – 40 milliseconds, it begins to be perceived as echo, making the sound more difficult to interpret, which is the opposite of what a hearing aid should be doing. 30 – 40 milliseconds is a much tighter latency than most existing A2DP solutions can provide, so this introduced a new requirement of very low latency. A 30 – 40 ms delay in the ambient sound equates to around 10 metres, which means that this is not an issue in most domestic rooms. It’s also well within the brain’s ability to cope with lip-synch, so video and audio will appear to be synchronized without the TV needing to adjust its video rendering delay.

Although the bandwidth requirements for hearing aids are relatively modest, with a bandwidth of 7kHz sufficient for mono speech and 11kHz for stereo music, these could not be easily met with the existing Bluetooth codecs whilst achieving that latency requirement. That led to a separate investigation to scope the performance requirements for a suitable codec, leading to the incorporation of the LC3 codec, which we’ll cover in Chapter 5.

2.1.1.3 Adding more users

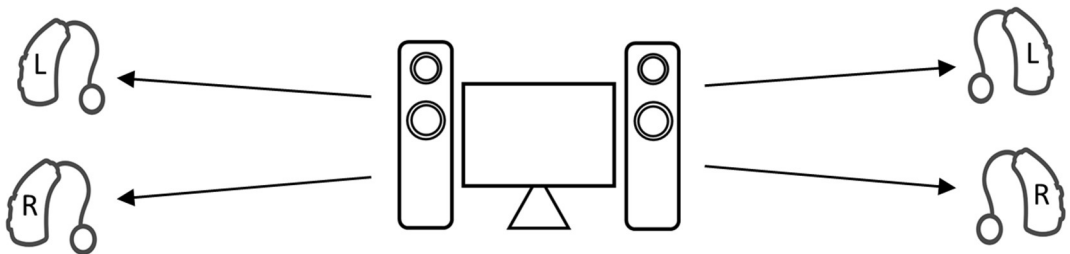


Figure 2.3 Adding in multiple listeners

Hearing loss may run in families, and is often linked with age, so it’s common for there to be more than one person in a household who wears hearing aids. Therefore, the new topology needed to support multiple hearing aid wearers. Figure 2.3 illustrates that use case for two people, both of whom should experience the same latency.

2.1.1.4 Adding more listeners to support larger areas

The topology should also be scalable, so that multiple people can listen, as in a classroom, conference centre, theatre or care home. That requirement positions Bluetooth LE Audio as

Section 2.1 - The use cases

a complementary solution for today's telecoil induction loops, offering more features and simpler installation. This required a Bluetooth Broadcast Transmitter which could broadcast mono or stereo audio streams capable of being received by any number of hearing aids which are within range, as shown in Figure 2.4.



Figure 2.4 A broadcast topology to replace telecoil infrastructure

Figure 2.4 also recognises the fact that some people have hearing loss in only one ear, whereas others have hearing loss in both, (which may often be different levels of hearing loss). That means that it should be possible to broadcast stereo signals at the same time as mono signals. It also highlights the fact that a user may wear hearing aids from two different companies to cope with those differences, or a hearing aid in one ear and a consumer earbud in the other.

2.1.1.5 Coordinating left and right hearing aids

Whatever the combination, it should be possible to treat a pair of hearing aids as a single set of devices, so that both connect to the same audio source and common features like volume control work on both of them in a consistent manner. This introduced the concept of coordination, where different devices which may come from different manufacturers would accept control commands at the same time and interpret them in the same way.

2.1.1.6 Help with finding broadcasts and encryption

With telecoil loops, users have only one option to obtain a signal - turn their telecoil receiver on, which picks up audio from the induction loop that surrounds them, or turn it off. Only one telecoil signal can be present in an area, so you don't have to choose which signal you want. On the other hand, that means you can't do things like broadcast multiple languages at the same time.

With Bluetooth, multiple Broadcast Transmitters can operate in the same area. That has obvious advantages, but introduces two new problems - how do you pick up the right audio stream, and how do you prevent others from listening in to a private conversation?

To help choose the correct stream, it's important that users can find out information about which audio streams that are available, so they can jump straight to their preferred choice. The richness of that experience will obviously differ depending upon how the search for the broadcast streams is implemented, either on the hearing aid, or on a phone or remote control, but the specification needs to cover all of those possibilities. Many public broadcasts wouldn't need to be private as they reinforce public audio announcements, such as when the next bus is due. However, others do need privacy. In environments like the home, you wouldn't want to pick up your neighbour's TV audio. Therefore, it is important that the audio streams can be encrypted, requiring the ability to distribute encryption keys to authorised users. That process has to be secure, but easy to do.

As well as the low latency, emulating current hearing aid usage added some other constraints. Where the user wears two hearing aids, regardless of whether they are receiving the same mono, or stereo audio streams, they need to render the audio within 25µs of each other to ensure that the audio image stays stable. That's equally true for stereo earbuds, but is challenging when the left and right devices may come from different manufacturers. It's worth stressing that this is microseconds, not milliseconds, requiring a level of synchronization that is more stringent than most audio developers are used to.

2.1.1.7 Practical requirements

Hearing aids are very small, which means they have very limited space for buttons. They are worn by all ages, but some older wearers have limited manual dexterity, so it's important that controls for adjusting volume and making connections can be implemented on other devices, which are easier to use. That may be the audio source, typically the user's phone, but it's common for hearing aid users to have small keyfob like remote controls as well. These have the advantage that they work instantly. If you want to reduce the volume of your hearing aid, you just press the volume or mute button; you don't need to enable the phone, find the hearing aid app and control it from there. That can take too long and is not a user experience that most hearing aid users appreciate. They need a volume and mute control method which is quick and convenient, otherwise they'll take their hearing aid out of their ear, which is not the desired behaviour.

There is another hearing aid requirement around volume, which is that the volume level (actually the gain) should be implemented on the hearing aid. The rationale for this is if the audio streams are transmitted at line level⁴ you get the maximum dynamic range to work with. For a hearing aid, which is processing the sound, it is important that the incoming signal

⁴ Line level is an audio engineering term referring to a standardised output level which is fed into an amplifier. In its general sense, it refers to a signal which has been set so that maximum volume of the audio volume would correspond to the full range of the output signal, i.e., it makes full use of the available dynamic range. If you've ever looked at a display of the audio output on a mixing desk, it's where the needle is hovering around the point where it enters the red zone.

Section 2.1 - The use cases

provides the best possible signal to noise ratio, particularly if it is being mixed with an audio stream from ambient microphones. If the gain of the audio is reduced at the source, it results in a worse signal to noise ratio.

An important difference between hearing aids and earbuds or headphones is that hearing aids are worn most of the time and are constantly active, amplifying and adapting the ambient sound to help the wearer hear more clearly. Users don't regularly take them off and pop them back in a charging case. The typical time a pair of hearing aids is worn each day is around nine and a half hours, although some users may wear them for fifteen hours or more. That's very different from earbuds and headphones, which are only worn when the user is about to make or take a phone call, or listen to audio. Earbud manufacturers have been very clever with the design of their charging cases to encourage users to regularly recharge their earbuds during the course of a day, giving the impression of a much greater battery life. Hearing aids don't have that option, so designers need to do everything they can to minimise power consumption.

One of the things that takes up power is looking for other devices and maintaining background connections. Earbuds get clear signals about when to do this – it's when they're taken out of the charging box and placed in the ear. Most also contain optical sensors to detect when they are in the ear, so they can go back to sleep if they're on your desk. Hearing aids don't get the same, unambiguous signal to start a Bluetooth connection as they're always on, constantly working as hearing aids. That implies that they need to maintain ongoing Bluetooth connections with other devices while they're waiting for something to happen. These can be low duty-cycle connections, but not too low, otherwise the hearing aid might miss an incoming call, or take too long to respond to a music streaming app being started. Because a hearing aid may connect to multiple different devices, for example a TV, a phone, or even a doorbell, connections like this would drain too much power, so there was a requirement for a new mechanism to allow them to make fast connections with a range of different products, without killing the battery life, and without maintaining constant connections. We'll explore this in more detail in Section 3.7 which looks at the new unconnected topology model.

2.1.2 Doing everything that HFP and A2DP can do

As the consumer electronics industry began to recognise the potential of the Bluetooth LE Audio features, which addressed many of the problems they had identified over the years, they made a pragmatic request for a second round of requirements, to ensure that Bluetooth LE Audio would be able to do everything that A2DP and HFP could do. They made the point that nobody would want to use Bluetooth LE Audio instead of Bluetooth Classic Audio if the user experience was worse.

These requirements increased the performance requirements on the new codec and introduced a far more complex set of requirements for media and telephony control. The original hearing aid requirements included quite limited control functionality for interactions with phones, assuming that most users would directly control the more complex features on their phone or TV, not least because hearing aids have such a limited user interface. Many consumer audio

products are larger, so don't have that limitation. As a result, new telephony and media control requirements were added to allow much more sophisticated control.

2.1.3 Evolving beyond HFP and A2DP

The third round of requirements was a reflection that audio and telephony applications have outpaced HFP and A2DP. Many calls today are VoIP⁵ and it's common to have multiple different calls arriving on a single device – whether that's a laptop, tablet or phone. Bluetooth technology needed a better way of handling calls from multiple different bearers. Similarly, A2DP hadn't anticipated streaming, and the search requirements that come with it, as it was written at a time when users owned local copies of music and rarely did anything more complex than selecting local files. Today, products needed much more sophisticated media control. They also needed to be able to support voice commands without interrupting a music stream.

The complexity in today's phone and conferencing apps where users handle multiple types of call, along with audio streaming, means that they make more frequent transitions between devices and applications. The inherent difference in architecture between HFP and A2DP has always made that difficult, resulting in a set of best practice rules which make up the Multi-Profile specification (MPS) for HFP and A2DP. However, the MPS is essentially a sticking plaster for just two use cases. The new Bluetooth LE Audio architecture was going to have to go beyond that and incorporate multi-profile support by design for an ever expanding number of use cases, with robust and interoperable transitions between devices and applications, as well as between unicast and broadcast. This also resulted in the removal of the complex role switch, which adds complexity within Bluetooth Classic applications.

As more people in the consumer space started to understand how telecoil and the broadcast features of hearing aids worked, they began to realise that broadcast might have some very interesting mass consumer applications. At the top of their list was the realisation that it could be used for sharing music. That could be friends sharing music from their phones, silent discos or “silent” background music in coffee shops and public spaces. Public broadcast installations, such as those designed to provide travel information for hearing aid wearers, would now be accessible to everyone with a Bluetooth headset. The concept of Audio Sharing, which we'll examine in more detail in Chapters 11 and 12, was born.

Potential new use cases started to proliferate. If we could synchronize stereo channels for two earbuds, why not for surround sound? Companies were keen to make sure that Bluetooth LE Audio supported smart watches and wristbands, which could act as remote controls, or even be audio sources with embedded MP3 players. The low latencies were exciting for the gaming

⁵ Voice over Internet Protocol, which is used for voice in most PC and mobile OTT (Over The Top) telephony applications.

Section 2.1 - The use cases

community. Microwave ovens could tell you when your dinner's cooked (you can tell that idea came from an engineer). The number of use cases continued to grow as companies saw how it could benefit their customers, their product strategies and affect the future use of voice and music.

The number of features has meant it has taken a long time to complete the specification. What is gratifying is that most of the new use cases which have been raised in the last few years have not needed us to go back and reopen the specifications – we've found that they were already supported by the architecture that had been defined. That suggests the Bluetooth LE Audio standards have been well designed and are able to support the audio applications we have today as well as new audio applications which are yet to come.

2.1.4 Addressing future audio applications

The specification process hasn't stopped. The release of the Bluetooth LE Audio specifications means that it's now possible to design and manufacture a very wide range of innovative products, but the work is continuing. Over the next few years, we will see more specifications being adopted, but for the moment, they're still confidential. If you're an Associate or Promoter member of the Bluetooth SIG, you can follow this work by joining the Hearing Aid, Generic Audio or the Audio, Telephony and Automotive Working Groups. Adopter members will get early access to the specifications when they are ready for prototyping.

2.1.5 Core requirements to support the hearing aid use cases

Having defined the requirements for topology and connections, it became obvious that a significant number of new features needed to be added to the Core specification to support them. This led to an additional round of work to determine how best to meet these new requirements in the Core.

The first part of the process was an analysis of whether the new features could be supported by extending the existing Bluetooth audio specifications rather than introducing a new audio streaming capability into Bluetooth LE. If that were possible, it would have provided backwards compatibility with current audio profiles. The conclusion, similar to an analysis that had been performed when Bluetooth LE was first developed, was that it would involve too many compromises and that it would be better to do a "clean sheet" design on top of Bluetooth® Core 4.1.

The proposal for the Bluetooth® Core Specification was to implement a new feature called Isochronous⁶ Channels which could carry audio streams in Bluetooth LE, alongside an existing

⁶ Isochronous means a sequence of events which are repeated at equal time intervals.

ACL⁷ channel. The ACL channel would be used to configure, set up and control the streams, as well as carrying more generic control information, such as volume, telephony and media control. The Isochronous Channels could support unidirectional or bidirectional Audio Streams, and multiple Isochronous Channels could be set up with multiple devices. This separated out the audio data and control planes, which makes Bluetooth LE Audio far more flexible.

It was important that the audio connections were robust, which meant they needed to support multiple retransmissions, to cope with the fact that some transmissions might suffer from interference. For unicast streams, there is an ACK/NACK acknowledgement scheme, so that retransmissions could stop once the transmitter knew that data had been received. For broadcast, where there is no feedback, the source would need to unconditionally retransmit the audio packets. During the investigation of robustness, it became apparent that the frequency hopping scheme⁸ used to protect LE devices against interference could be improved, so that was added as another requirement.

Broadcast required some new concepts, particularly in terms of how devices could find a broadcast without the presence of a connection. Bluetooth LE uses advertisements to let devices announce their presence. Devices wanting to make a connection scan for these advertisements, then connect to the device they discover to obtain the details of what it supports, how to connect – including information on when it's transmitting, what its hopping sequence is and what it does. That requires a lot more information than can be fitted into a normal Bluetooth LE advertisement. To overcome this limitation, the Core added a new feature of Extended Advertisements (EA) and Periodic Advertising trains (PA) which allow this information to be carried in data packets on general data channels which are not normally used for advertising. To accompany this, it added new procedures for a receiving device to use this information to determine where the broadcast audio streams were located and synchronize to them.

The requirement that an external device can help find a Broadcast stream added a requirement that it could then inform the receiver of how to connect to that stream – essentially an ability for the receiver to ask for directions from a remote control and be told where to go. That's accomplished by a Core feature called PAST – Periodic Advertising Synchronization Transfer, which is key to making broadcast acquisition simple. PAST is a really useful feature for hearing

⁷ ACL channels are Asynchronous Connection-oriented Logical transports which are used for control requests and responses in Bluetooth LE GATT transactions. They carry all of the control information in Bluetooth LE Audio (with the sole exception of Broadcast Control subevents) and form the control plane.

⁸ Bluetooth uses an adaptive frequency hopping (AFH) scheme to avoid interference, constantly changing the frequency channel a device uses to transmit data, based on an analysis of current interference. The sequence of channels to use is called the hopping scheme, which needs to be conveyed from the Central Device to all of its Peripherals.

Section 2.2 - The Bluetooth LE Audio architecture

aids, as scanning takes a lot of power. Minimizing that scanning, by offloading it to another device, is a useful approach to help prolong the battery life of a hearing aid.

The hearing aid requirements also resulted in a few other features being added to the core requirements, primarily around performance and power saving. The first was an ability for the new codec to be implemented in the Host or in the Controller. The latter makes it easier for hardware implementations, which are generally more power efficient. The second was to put constraints on the maximum time a transmission or reception needed to last, which impacted the design of the packet structure within Isochronous Channels. The reason behind this is that many hearing aids use primary, zinc-air batteries, because of their high power density. However, this battery chemistry relies on limiting current spikes and high-power current draw. Failing to observe these restrictions results in a very significant reduction of battery life. Meeting them shaped the overall design of the Isochronous Channels.

Two final additions to the Core requirements, which came in fairly late in the development, were the introduction of the Isochronous Adaptation Layer (ISOAL) and the Enhanced Attribute Protocol (EATT).

ISOAL allows devices to convert Service Data Units (SDUs) from the upper layer to differently sized Protocol Data Units (PDUs) at the Link Layer and vice versa. The reason this is needed is to cope with devices which may be using the recommended timing settings for the new LC3 codec, which is optimised for 10ms frames, alongside connections to older Bluetooth devices which run at a 7.5ms timing interval.

EATT is an enhancement to the standard Attribute Protocol (ATT) of Bluetooth LE to allow multiple instances of the ATT protocol to run concurrently. This removes the blocking limitation of ATT, where each command needs to be completed before the next one can be actioned.

The Extended Advertising features were adopted in the Bluetooth® Core 5.0 release. This was enhanced with Periodic Advertising in Bluetooth® Core 5.1, and Isochronous Channels, EATT and ISOAL in the Bluetooth® Core 5.2 release. This provided the foundation on which all of the other Bluetooth LE Audio specifications have been built. Three further revisions of the Bluetooth® Core Specification – Bluetooth® Core 5.3, 5.4 and 6.0 have been released, which contain minor updates to the Isochronous Channels feature, but no significant changes have been made.

2.2 The Bluetooth LE Audio architecture

The Bluetooth LE Audio architecture has been built up in layers, as has every other Bluetooth specification before it. This is illustrated in Figure 2.5, which shows the main new specification blocks relating to Bluetooth LE Audio, (with key existing ones greyed out or dotted).

At the bottom we have the Core, which contains the radio and Link Layer, (collectively known

as the Controller). It is responsible for sending Bluetooth packets over the air. On top of that is the Host, which has the task of telling the Core what to do for any specific application. The separation between the Controller and Host is historic, reflecting the days when a Bluetooth radio would be sold in a USB stick or a PCMCIA card, with the Host implemented as a software application on a PC. Today both Host and Controller are often integrated into a single chip. In a phone, much of the stack may be implemented in the phone’s operating system, with the manufacturer exposing features to app developers through their own APIs.

In the Host, there is a new structure called the Generic Audio Framework or GAF. This is an audio middleware, which contains all of the functions which are considered to be generic, i.e., features which are likely to be used by more than one audio application. The Core and the GAF are the heart of Bluetooth LE Audio. They provide great flexibility. Finally, at the top of the stack, we have what are termed “top level” profiles, which add application specific information to the GAF specifications.

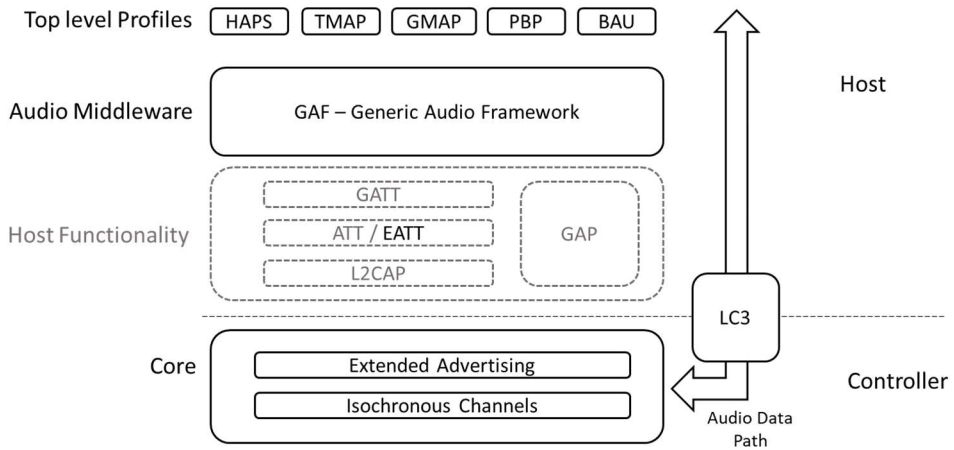


Figure 2.5 The Bluetooth® LE Audio architecture

It is perfectly possible to build interoperable Bluetooth LE Audio applications using just the GAF specifications. The individual specifications within it have been defined to ensure a base level of interoperability, which would enable any two Bluetooth LE Audio devices to transfer audio between them. The top level profile specifications largely add features specific to a particular type of audio application, mandating features which GAF only defines as optional, and adding application-specific functionality. The intention is that the top level profiles are relatively simple, building on features from within the GAF. In most cases, they can be considered as predominantly configuration requirements.

At first glance, the Bluetooth LE Audio architecture looks complex, as we’ve ended up with eighteen different specifications inside the Generic Audio Framework, as well as an expanded Core, the new LC3 codec and a growing number of top level profiles. But there’s a logic to this. Each specification is trying to encapsulate the specific elements of the way you set up and

Section 2.2 - The Bluetooth LE Audio architecture

control different aspects of an audio stream. In the remainder of this chapter, I'll briefly explain each one and how they fit together. Then, in the rest of the book, we'll look at how each individual specification works and how they interact.

2.2.1 Profiles and Services

All of the specifications within the GAF are classified as Profiles or Services using the standard Bluetooth LE GATT model depicted in Figure 2.6.

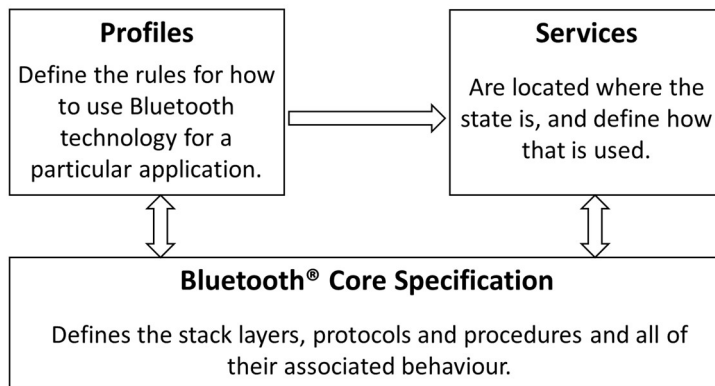


Figure 2.6 The Bluetooth® LE Profile and Service model

In Bluetooth LE, Profiles and Services can be considered as Clients and Servers. The Service is implemented where the state⁹ resides, whereas Profile specifications describe how the state behaves and includes procedures to manage it. Service specifications define one or more characteristics which can represent individual features or the states of a state machine. They can also be control points, which cause transitions between the states of a state machine. Profiles act on these characteristics, reading or writing them and being notified whenever the values change. Multiple devices, each acting as Clients, can operate on a Server.

Traditionally, in classic Bluetooth profiles (which didn't have a corresponding service), there was a simple one-to-one relationship with only one Client and one Server, with everything described in a Profile specification. In Bluetooth LE Audio, the one-to-many topology is much more common, particularly in features like volume control and the selection of a Broadcast Source, where a user may have multiple devices implementing the Profile specification, acting as Clients. In most cases, these act on a first come, first served basis.

The number of different control profiles that can be used in Bluetooth LE Audio drove the EATT enhancement to the Core. Profiles and Services communicate using the Attribute Protocol (ATT), but ATT assumes that only one command is occurring at a time. If more

⁹ State refers to the value of a parameter or the current position within a state machine.

than one is happening, the second command can be delayed, because ATT is a blocking protocol. To get around this, the Extended Attribute Protocol (EATT) was added in the Bluetooth® Core 5.2 release, allowing multiple ATT instances to operate at the same time.

Figure 2.7 provides an overview of the Bluetooth LE Audio architecture, putting a name, or more precisely, a set of letters, to all of the 18 specifications which make up the GAF, along with the six in the current top level profiles. The dotted boxes indicate sets of profiles and services which work together. In most cases there is a one-to-one relationship of a profile and a service, although in the case of the Basic Audio Profile (BAP)¹⁰ and the Voice Control Profile (VCP), one profile can operate on three different services. The Public Broadcast Profile (PBP) and the Broadcast Audio URI (BAU) are anomalies, as neither has a service, but that's one of the consequences of broadcast, as you cannot have a traditional Client-Server interaction when there is no connection.

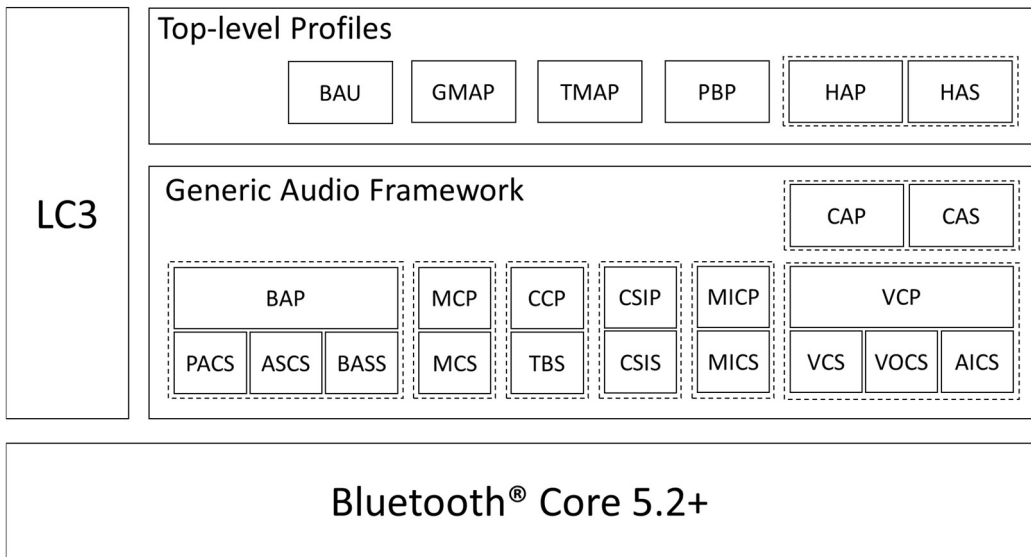


Figure 2.7 Overview of the Bluetooth LE Audio Specifications

2.2.2 The Generic Audio Framework

We can now look at the constituent parts of the GAF. There is a significant amount of interaction between the various specifications, which makes it difficult to draw a clear hierarchy or set of relationships between them, but they can be broadly separated into four

¹⁰ If the acronym or initialism ends with a “P” it’s a profile. If it ends with an “S”, it’s a service. If it ends with PS, it normally refers to a combination of separate Profile and Service documents. For example, BAPS is a shorthand acronym that refers to the combination of the Basic Audio Profile, the Audio Stream Control Service and the Published Audio Capabilities Service and the Broadcast Audio Stream Service.

Section 2.2 - The Bluetooth LE Audio architecture

functional groups, arranged as in Figure 2.8.

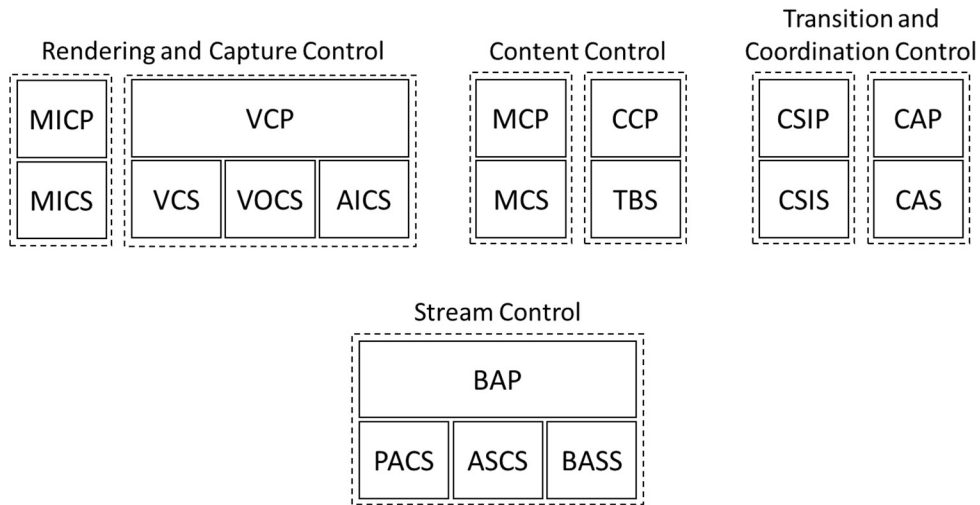


Figure 2.8 Functional grouping of specifications within the Generic Audio Framework

This grouping is largely for the sake of explanation. In real implementations of Bluetooth LE Audio, most of these specifications interact with each other to a greater or lesser degree. It's perfectly possible to make working products with just a few of them, but to design richly featured, interoperable products, the majority of them will be required.

2.2.3 Stream configuration and management – BAPS

Starting at the bottom of Figure 2.8, we have a group of four specifications which are collectively known as the BAPS specifications. These four specifications form the foundation of the Generic Audio Framework. At their core is BAP – the Basic Audio Profile, which is used to set up and manage unicast and broadcast Audio Streams. As a profile, it works with three services:

- PACS – the Published Audio Capabilities Service, which exposes the capabilities of a device,
- ASCS - the Audio Stream Control Service, which defines the state machine for setting up and maintaining unicast audio streams, and
- BASS - the Broadcast Audio Scan Service which defines the procedures for discovering and connecting to broadcast audio streams and distributing the broadcast encryption keys.

Between them, they are responsible for the way the underlying Isochronous Channels which carry the audio data are set up. They also define a standard set of codec configurations for LC3 and a corresponding range of Quality of Service (QoS) settings for use with broadcast and unicast applications.

State machines for each individual Isochronous Channel are defined for both unicast and broadcast, both of which move the audio stream through a configured state to a streaming state, as illustrated in the simplified state machine of Figure.2.9.

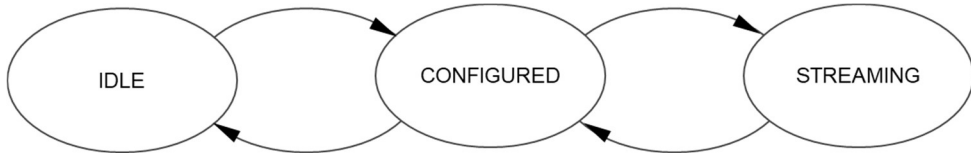


Figure.2.9. The simplified Isochronous Channel state machine

For unicast, the state machine is defined in the ASCS specification. The state resides within individual audio endpoints in the Server, with the Client control defined in BAP. For broadcast, where there is no connection between the transmitter and receiver, the concept of a Client-Server model becomes a little tenuous. As a result, BAP only defines a state machine for the transmitter, and it is solely under the control of its local application. With broadcast, the receiver needs to detect the presence of a stream and then receive it, but has no way of affecting its state.

Multiple unicast or broadcast Isochronous Channels are bound together in Groups (which we explore in Chapter 4). BAP defines how these groups, and their constituent Isochronous Channels are put together for both broadcast and unicast streams.

You can make a Bluetooth LE Audio product with just three of these specifications; BAP, ASCS and PACS for unicast and BAP alone for broadcast (although you'll need to add PACS and BASS if you want to use a phone or remote control to help find the broadcasts). It would be quite a limited device in terms of functionality – just setting up an audio stream, using it to transmit audio and stopping it. However, by being able to do this, the BAPS set of specifications provide a base level of interoperability for all Bluetooth LE Audio devices. If two Bluetooth LE Audio devices have different top level profiles, they should still be able to set up an audio stream using BAP. It may have restricted functionality, but should provide an acceptable level of performance, removing the issue of multi-profile incompatibility that is present in Bluetooth Classic Audio, where devices with no common audio profile would not work together.

2.2.4 Rendering and capture control

Having set up a stream, users want to control the volume, both of the audio streams being rendered in their ears and the pick-up of microphones.

Volume is a surprisingly difficult topic, as there are multiple places where the volume can be adjusted – on the source device, on the hearing aid, earbud or speaker, or on another “remote control” device, which could be a smartwatch or a separate Controller. In Bluetooth LE

Section 2.2 - The Bluetooth LE Audio architecture

Audio, the final gain of the volume is performed in the hearing aid, earbud or speaker; not on the incoming audio stream prior to encoding. With that assumption, the Volume Control Profile (VCP) defines how a Client manages the gain on the Audio Sink device. The state of that gain is defined in the Volume Control Service (VCS), with one instance of VCS on each audio sink. The Volume can be expressed as absolute or relative, and can also be muted.

Where multiple Bluetooth LE Audio Streams are being sent from an Initiator, as is the case with earbuds, hearing aids and speakers, a second service is required. VOCS - the Volume Offset Control Service, effectively acts as a balance control¹¹, allowing the relative volume of multiple devices to be adjusted. These may be rendered on different devices, such as separate left and right earbuds or speakers, or on a single device, like a pair of headphones or a soundbar. VOCS is orthogonal to the audio streaming – it doesn't know whether the audio content is the same for multiple streams, or is different. It's only purpose is to set the relative gain across multiple rendering devices.

The Audio Input Control Service (AICS) acknowledges the fact that many devices now have the ability to support a number of different audio streams. AICS lets you control multiple different inputs that can be mixed together and rendered within your earbud or speaker. Figure 2.10 illustrates how these three services could be used in a soundbar which has a Bluetooth, HDMI and microphone input.

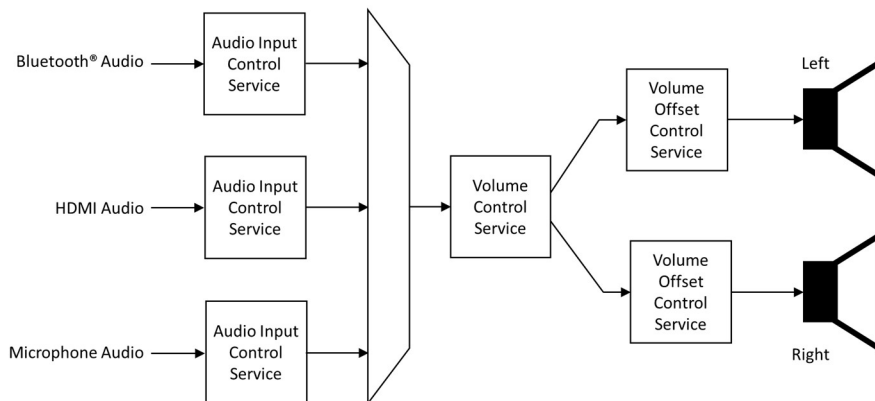


Figure 2.10 *Audio Input Control Service (AICS), Volume Control Service (VCS) and Volume Offset Control Service (VOCS)*

For a hearing aid, the inputs might be a Bluetooth stream, a microphone providing an ambient audio stream, and a telecoil antenna receiving a stream from an audio loop. At any point in time, the wearer may want to hear a combination of these different inputs. AICS supports

¹¹ “Balance” is the usual consumer nomenclature. Audio engineers would probably consider it to be a mixing control.

that flexibility.

An important feature of the volume services is that they notify any changes back to Client devices running the Voice Control Profile. This ensures that all potential Volume Controllers are kept up to date with any changes to their state, regardless of whether that occurred over a Bluetooth link or from a local volume control on an Audio Sink. This ensures that they all have a synchronized knowledge of the volume state, so that the user can make changes from any one of them, without any unexpected effects from their having an outdated knowledge of the current state of the volume level.

A complementary pair of specifications, MICP and MICS, the Microphone Control Profile and Service, are responsible for controlling the microphones that reside within hearing aids and earbuds. Nowadays, these devices typically contain multiple microphones. Hearing aids listen to both ambient sound (their primary function), as well as audio that's being received over Bluetooth. As earbuds get more sophisticated, we're increasingly seeing similar ambient sound capabilities being built into them, with a growing popularity for some degree of “transparency”, where ambient sound is mixed with the Bluetooth audio stream.

MICP works in conjunction with AICS and MICS to control the overall gain and muting of multiple microphones. These specifications are normally used for control of the captured audio that is destined for a Bluetooth stream, but can be used more widely. Figure 2.11 illustrates the use of these services.

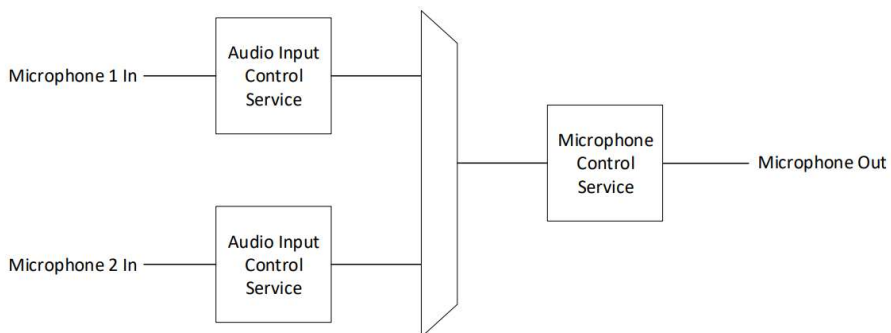


Figure 2.11 The Audio Input Control Service (AICS) used with the Microphone Control Service (MICS)

2.2.5 Content control

Having specified how streams are set up and managed and how volume and microphone input is handled, we come to content control. The content we listen to is generated outside of the Bluetooth specifications – it may be streaming music, live TV, a phone call or a video conference. What the content control specifications do is to allow control in terms of starting, stopping, answering, pausing and selecting the streams. These are the types of control which were embedded into HFP and the Audio/Video Remote Control Profile (AVRCP), which

Section 2.2 - The Bluetooth LE Audio architecture

accompanies A2DP. In Bluetooth LE Audio they are separated out into two sets of specifications – one for telephony in all of its forms, and the other for media. The key differentiator is that telephony is about the state of the call or calls, which normally reflects the state of the telephony service, whereas media control acts on the state of the stream – when and how it’s played and how it’s selected. Because these are decoupled from the audio streams, they can now be used to help control transitions, such as pausing music playback when you accept a phone call and restoring it when the call is finished. For both of these pairs of specifications, the Service resides on the primary audio source – typically the phone, PC, tablet or TV, whereas the Profile is implemented on the device which receives the audio stream, such as the hearing aid or earbud. As with the rendering and capturing controls, multiple devices can act as Clients, so telephony and media state could be controlled from a smartwatch instead of, or as well as from an earbud.

The Media Control Service (MCS) resides on the source of audio media and reflects the state of the audio stream. The state machine allows a Client using the Media Control Profile (MCP) to transition each media source through Playing, Paused and Seeking states. At its simplest, it allows an earbud to control Play and Stop. However, MCS goes far beyond that, providing all of the features which users expect from content players today. It also provides higher level functions, where a user can search for tracks, modify the playing order, set up groups and adjust the playback speed. It defines metadata structures which can be used to identify the tracks and uses the existing Object Transfer Service (OTS) to allow a Client to perform media searches on the Server, or more typically the application behind it. All of this means that a suitably complex device running the Media Control Profile can recreate the controls of a music player.

Telephony control is handled in a similar way using the Telephone Bearer Service (TBS), which resides on the device involved in the call (typically the phone, PC or laptop), with the complementary Call Control Profile (CCP) controlling the call by writing to the state machine in the TBS instance. TBS and CCP have expanded past the limitations of Hands-Free Profile to reflect the fact that we now use telephony in many different forms. It's no longer focused on traditional circuit switched and cellular bearers, but includes full support for PC and web-based communication and conferencing applications, using multiple, different types of bearer service. TBS exposes the state of the call using a generic state machine. It supports multiple calls, call handling and joining, caller ID, inband and out of band ringtone selection and exposes call information, such as signal strength.

Both TBS and MCS acknowledge the fact that there may be multiple sources of media and multiple different call applications on the Server devices. To accommodate this, both can be instantiated multiple times – once for each instance of an application. This allows a Client with the complementary profile to control each application separately. Alternatively, a single instance of the service can be used, with the media or call device using its specific implementation to direct the profile commands to the correct application. The single instance variants of TBS and MCS are known as the Generic Telephone Bearer Service (GTBS) and

Generic Media Control Service (GMCS) and are included in the TBS and MCS specifications respectively. We'll look at these in more detail in Chapter 9.

2.2.6 Transition and coordination control

Next, we come to the Transition and Coordination Control specifications. Their purpose is to glue the other specifications together, providing a way for the top level profiles to call down to them without having to concern themselves with the fine detail of setting things up.

One of the major enhancements in Isochronous Channels is the ability to stream audio to multiple different devices and render it at exactly the same time. The most common application of this is in streaming stereo music to left and right earbuds, speakers or hearing aids. The topology and synchronisation of rendering are handled in the Core and BAP, but ensuring that control operations occur together, whether that's changing volume or transitioning between connections is not. That's where the Coordinated Set Identification Profile (CSIP) and Coordinated Set Identification Service (CSIS) come in.

Where two or more Bluetooth LE Audio devices are expected to be used together, they are called a Coordinated Set and can be associated with each other by use of the Coordinated Set Identification Service. This allows other profiles, in particular CAP – the Common Audio Profile, to treat them as a single entity. It introduces the concepts of Lock and Rank to ensure that when there is a transition between audio connections, whether that's to a new unicast or broadcast stream, the members of the set always react together. This prevents a new connection only being applied to a subset of the devices in the set, such as a TV connecting to your right earbud, while your phone connects to your left. Devices designed to be members of Coordinated Sets are generally configured as set members during manufacture.

Multiple devices which are not configured as members of a Coordinated Set can still be used in GAF as an ad-hoc set. In this case they need to be individually configured by the application. It means that they do not benefit from the locking feature of CSIS, which could result in different connections to members of the ad-hoc set.

CAP – the Common Audio Profile, introduces the Commander role, which brings together the features which can be used for remote control of Bluetooth LE Audio Streams. The Commander is a major change from anything we've seen in previous Bluetooth specifications, as it allows an additional device to be used to affect the audio experience, leading to the design of ubiquitous, distributed remote control for audio. It is also particularly useful for encrypted broadcasts, where, combined with a Broadcast Transmitter, it provides a way to deliver a private listening experience. We'll explore that in more detail in Chapters 8 and 12.

CAP uses CSIS and CSIP to tie devices together and ensure that procedures are applied to both. It also introduces the concept of Context Types and Content Control IDs which allow applications to make decisions about stream setup and control based on a knowledge of the controlling devices, the use cases for the audio data and which applications are available. This

Section 2.3 - Talking about Bluetooth LE Audio

is used to inform transitions between different streams, whether that is prompted by different applications on a device or a request from a different device for an audio connection. A lot of this functionality is based on new concepts, which have been introduced in Bluetooth LE Audio. These are explained in more detail in Chapter 3.

2.2.7 Top level Profiles

Finally, on top of the GAF specifications, we have top level profiles which provide additional requirements for specific audio use cases. Currently, these are the Hearing Access Profile and Service (HAP and HAS), which cover applications for the hearing aid ecosystem; the Telephony and Media Audio Profile (TMAP)¹², which specifies the use of higher quality codec settings and more complex media and telephony control, along with the Gaming Audio Profile (GMAP), which is aimed at low latency audio for gaming. The Public Broadcast Profile (PBP), which helps users select globally interoperable broadcast streams, while the Broadcast Audio URI (BAU) defines how out of band methods like QR codes can be used to assist Broadcast Assistants in finding broadcasts. The PBP and BAU specifications are anomalous in having no accompanying service, but that's a consequence of the nature of broadcasts, where there is no connection for any Client-Server interaction.

2.2.8 The Low Complexity Communications Codec (LC3)

Although not a part of the GAF, the Bluetooth LE Audio specifications include a new, efficient codec called LC3 which is the mandated codec for Bluetooth LE Audio streams. This provides excellent performance for telephony speech, wideband and super-wideband speech and high-quality audio and is the mandated codec within BAP. Every Bluetooth LE Audio product has to support the LC3 codec to ensure interoperability, but additional and proprietary codecs can be added if manufacturers require. LC3 encodes audio into single streams, so stereo is encoded as separate left and right streams. This means that GAF can configure a unicast stream to an earbud to carry only the audio which that earbud requires. A Broadcast Transmitter sending music would normally include individual left and right Audio Streams in its broadcast. Individual devices would only need to receive and decode the data relevant to the stream which they want to render.

2.3 Talking about Bluetooth LE Audio

Writing over twenty specifications has generated a lot of new abbreviations, including ones for the name of each individual specification. Over time, the working groups have come up with different ways of referring to these – sometimes as acronyms (where you pronounce them as words), sometimes as initialisms (where you just pronounce the letters, and occasionally as a mix of both. The following table captures the current pronunciations of the most commonly

¹² There are corresponding, minimalist TMAP and GMAP specifications, which are included within the TMAP and GMAP specifications.

used ones, to help you understand anyone talking about Bluetooth LE Audio.

Section 2.3 - Talking about Bluetooth LE Audio

2.3.1 Acronyms

The following abbreviations are all pronounced as words, or a combination of letter and word:

Abbreviation	Meaning	Pronunciation
AICS	Audio Input Control Service	<i>aches (as in aches and pains)</i>
ASE	Audio Stream Endpoint	<i>ase (rhymes with case)</i>
ATT	Attribute Protocol	<i>at</i>
BAP	Basic Audio Profile	<i>bap (rhymes with tap)</i>
BAPS	The set of BAP, ASCS, BASS and PACS	<i>baps</i>
BASE	Broadcast Audio Source Endpoint	<i>base (rhymes with case)</i>
BAIRF	Broadcast Immediate Rendering Flag	<i>barf</i>
BASS	Broadcast Audio Scan Service	<i>rhymes with mass, not mace</i>
BAU	Broadcast Audio URI	<i>bow (rhymes with cow)</i>
BIG	Broadcast Isochronous Group	<i>big</i>
BIS	Broadcast Isochronous Stream	<i>biss</i>
CAP	Common Audio Profile	<i>cap (rhymes with tap)</i>
CAS	Common Audio Service	<i>cas (rhymes with mass)</i>
CIG	Connected Isochronous Group	<i>sig</i>
CIS	Connected Isochronous Stream	<i>sis</i>
CSIP	Coordinated Set Identification Profile	<i>see - sip</i>
CSIS	Coordinated Set Identification Service	<i>see - sis</i>
CSIPS	The set of CSIP and CSIS	<i>see - sips</i>
EATT	Enhanced ATT	<i>ee - at</i>
GAF	Generic Audio Framework	<i>gaffe</i>
GAP	Generic Access Profile	<i>gap (rhymes with tap)</i>
GATT	Generic Attribute Profile	<i>gat (rhymes with cat)</i>
GMAP	Gaming Audio Profile	<i>jee-map</i>
GMAS	Gaming Audio Service	<i>jee-mass</i>
HAP	Hearing Access Profile	<i>bap (rhymes with tap)</i>
HARC	Hearing Aid Remote Controller	<i>hark</i>
HAS	Hearing Access Service	<i>bass (rhymes with mass)</i>
HAUC	Hearing Aid Unicast Client	<i>hawk</i>
INAP	Immediate Need for Audio related Peripheral	<i>eye - nap</i>
L2CAP	Logical Link Control and Adaptation protocol	<i>el - two - cap</i>
MICP	Microphone Control Profile	<i>mick - pee</i>
MICS	Microphone Control Service	<i>mick - ess</i>
PAC	Published Audio Capabilities	<i>pack</i>
PACS	Published Audio Capabilities Service	<i>packs</i>

Abbreviation	Meaning	Pronunciation
PAST	Periodic Advertising Synchronization Transfer	<i>past (rhymes with blast)</i>
PBAS	Public Broadcast Audio Stream Announcement	<i>pee - bass (bass rhymes with mass)</i>
PHY	physical layer	<i>fy (rhymes with fly)</i>
QoS	Quality of Service	<i>kwos</i>
RAP	Ready for Audio related Peripheral	<i>rap</i>
SIRK	Set Identity Resolving Key	<i>sirk (like the first syllable of circus)</i>
TMAP	Telephony and Media Audio Profile	<i>tee - map</i>
TMAS	Telephony and Media Audio Service	<i>tee - mas</i>
VOCS	Volume Offset Control Service	<i>vocks</i>

Table 2.1 Pronunciation guide for Bluetooth LE Audio acronyms

2.3.2 Initialisms

The rest are simply pronounced as the individual letters that make up the abbreviation, e.g., CCP is pronounced “*see – see – pee*” and LC3 is “*el – see – three*”.

The most common initialisms in Bluetooth LE Audio are: ACL, AD, ASCS, BMR, BMS, BR/EDR, CCID, CCP, CG, CSS, CT, CTKD, EA, FT, GMCS, GSS, GTBS, HA, HCI, IA, IAC, IAS, , IRC, IRK, LC3, MCP, MCS, MTU, NSE, PA, PBA, PBK, PBP, PBS, PDU, PTO, RFU, RTN, SDP, SDU, TBS, UI, UMR, UMS, UUID, VCP and VCS. These are all explained in the glossary.

2.3.3 Irregular pronunciations

OOB is an oddity, as it is almost always spoken in full, i.e., “*out of band*”, although the abbreviation is generally used in text.

-oOo-

That was a whistle-stop tour through the main parts of the Generic Audio Framework. We will see even more specifications appear in GAF in the future, but for now, the ones described above emulate and expand on the audio functionality that Bluetooth has today with the classic audio profiles.

In the remainder of the book, we’ll see how BAPS (BAP, BASS, ASCS and PACS) form the basis of everything that you do with Bluetooth LE Audio. The other specifications add usability and functionality, while CAP glues it all together. As a first step, we’ll look at some of the new concepts which have been necessary to address the Bluetooth LE Audio requirements.

Chapter 3. New concepts in Bluetooth® LE Audio

To provide designers with the flexibility they need, the new Bluetooth LE Audio specifications have introduced some important new concepts. In this chapter, we'll look at what they do and why they are needed. Because these features are closely integrated into the specifications, some of the descriptions below will become clearer as we dive down into the detail of the Core and the GAF. However, it's useful to introduce them at this stage, as they pervade much of what follows.

3.1 Multi-profile by design

As we've seen, one of the challenges faced by Bluetooth Classic Audio has been the multi-profile issue, where users change between streaming music, making phone calls and using voice recognition. As the complexity of audio use cases continues to increase, that problem won't improve. The two successful classic Bluetooth audio profiles – Hands-Free Profile (HFP) and music streaming (A2DP), were designed at a time where users were assumed to use one or the other, starting new, independent sessions as they moved from phone calls to music. It soon became apparent that these were not independent functions, but that phone users would expect one use case to interrupt the other. Over the years, the industry has developed rules and methods to address this problem, culminating in the publication of the Multi Profile Specification (MPS) in 2013, which essentially collects sets of rules for common transitions between these profiles.

The Multi Profile Specification is not particularly flexible and didn't foresee the emergence of new use cases, such as voice recognition, voice assistants and interruptions from GPS satnavs. Nor are the underlying specifications particularly versatile. Although HFP has gone through multiple versions and spawned around twenty complementary specifications addressing specific aspects of car to phone interaction, it still struggles to keep up with non-cellular VoIP telephony applications. Neither it, nor A2DP, have sufficiently addressed the changing nature of audio, where users connect to multiple different audio sources during the course of a day, and might also own multiple different headsets and earbuds.

From the start, Bluetooth LE Audio was developed with the philosophy that it would support “multi-profile by design”. It recognised that users might have multiple headsets and audio sources, constantly changing connections in an ad-hoc manner. The addition of broadcast audio makes this even more important, as the projected uptake of broadcast in venues, shops, leisure facilities and travel will extend the number of different connections a user is likely to make each day. It was obviously going to be important to provide tools to make the user experience seamless.

3.2 The Audio Sink led journey

Alongside these demands came a realization that the phone would not necessarily remain the centre of the audio universe, which had been a tacit assumption throughout the evolution of

the Bluetooth Classic Audio specifications. As an increasing number of different audio sources became available, it was important to consider how a user would connect their earbuds or hearing aids through the course of the day. This turned the perceived view of a phone-centric audio world on its head, replacing it with the concept of an Audio Sink¹³ led journey.

It's an approach where the phone is no longer the arbiter of what you listen to. Instead, it assumes that users are increasingly likely to wear a pair of hearing aids or earbuds throughout the day, constantly changing their audio source. It may start with an alarm clock, followed by asking your voice assistant to turn your radio on, travel updates as you wait at your bus stop, voice control for your computer, receiving phone calls, interruptions from the doorbell and relaxing in front of the TV when you get home, until the microwave tells you that your dinner is ready. This conceptual turn-around regarding who is in control takes a lot from the experience of hearing aid wearers, who typically wear their hearing aids for most of the day. For a large part of that time, the hearing aid works without any Bluetooth link, just listening to and reinforcing the sound around the wearer. But the wearer can connect it to a wide number of infrastructure audio sources – supermarket checkout machines, public transport information, theatres and auditoria and TVs, as well as phones and music players if they support Bluetooth Classic Audio. It's functionality that many more consumers are likely to use once it becomes widely available.

Phones will, of course, remain a major source of audio, but increasingly, so are PCs, laptops, TVs, tablets, Hi-Fi and voice assistants. They're also being joined by a range of smart home devices from doorbells to ovens. At the same time, consumers are buying more headsets. It's not uncommon for someone to own a pair of sleep buds, a set of earbuds, a pair of stereo headphones and a soundbar or Bluetooth speakers. As the potential combinations multiply, it's vital that the user interfaces can accommodate this range of different connections and the transitions between them.

3.3 Terminology

Different parts of the Bluetooth LE Audio specifications use different names for the transmitting and receiving devices and what they do. Each specification refers to these as Roles, so by the time we've gone from the Core to a top level profile we will have accumulated at least four different Roles for each device. There is a very good reason for this, as each step up the stack results in more specificity within those Roles, with the result that devices can implement specific combinations to fulfil a targeted function. But they can get confusing.

¹³ In this case, the Audio Sink is the general term for what you have in your ear to listen to audio streams. It sidesteps the fact that it can be an Audio Source as well. The point is that the device in your ear can now be more involved in decisions.

Section 3.3 - Terminology

The Core refers to Central and Peripheral devices, where Central devices send commands and Peripherals respond. In BAP they're called Clients and Server roles. Moving up a layer, CAP defines them as Initiator and Acceptor roles. The Initiator role exists in a Central device, which is responsible for setting up, scheduling and managing the Isochronous Streams. Acceptors are the devices that participate in these streams – typically hearing aids, speakers and earbuds. There is always one Initiator, but there can be multiple Acceptors. The different nomenclature is shown in Figure 3.1.

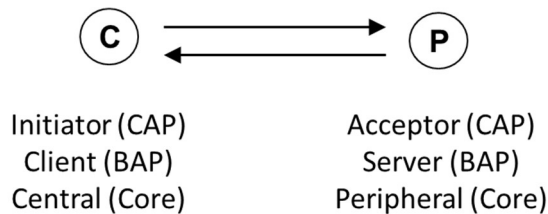


Figure 3.1 Bluetooth LE Audio roles

I'm going to use the Initiator and Acceptor names most of time (even though they're technically Roles), because I think they best explain the way that Bluetooth LE Audio works. Both Initiators and Acceptors can transmit and receive audio streams at the same time, and they can both contain multiple Bluetooth LE Audio Sinks and Sources. The important thing to remember is that the Initiator is the device that performs the configuration and scheduling of the Isochronous Streams, and the Acceptor is the device that accepts those streams. That concept applies both in unicast and broadcast. In broadcast, the specifications use the terms Broadcast Source and Broadcast Sink to refer to roles, but most developers had chosen to call them Broadcast Transmitters and Broadcast Receivers. As there is no link to allow them to affect each other, the name pairs of transmitter / source and receiver / sink have identical meanings, so at point they'll be used interchangeably.

At this point, it's worth having a slight diversion to look at some of the terminology which is used within Bluetooth LE Audio. This will introduce some features we haven't discussed yet, which we'll cover in detail in Chapter 4, when we get to Isochronous Streams.

Throughout the specifications, there are a lot of different names and phrases describing channels and streams. They have some very distinct and sometimes slightly different meanings depending on which of the specifications you're looking at, but I've tried to capture the main ones in Figure 3.2.

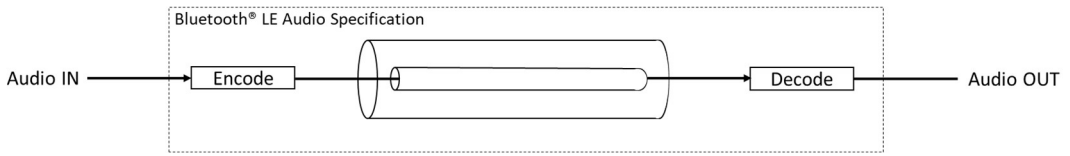


Figure 3.2 Bluetooth LE Audio terminology – an Audio Channel

The diagram shows the conceptual transmission of audio data from left to right. At the very left, we have audio coming in, which could be an analogue or digital signal. It's shown as Audio IN. At the other end we see audio being rendered at Audio OUT, which may be on a speaker or an earbud. The audio at these two points is called the Audio Channel (both ends are the same Audio Channel), which is equivalent to what would be carried if it were a wired connection between an MP3 player and a speaker. An Audio Channel is defined in BAP as a unidirectional flow of audio into the input of your Bluetooth device and then out of the other end. (In practice, that “out” will normally be directly into your headphone transducer or a speaker. Or it could remain digital for further processing stages.) The details of the input and output of the Audio Channel are implementation specific and outside the Bluetooth specifications. What the audio is and how the audio is handled after the wireless transmission and decoding is outside the scope of the specification¹⁴.

How that audio input is carried to the audio output is what is defined inside the Bluetooth specifications. It is represented by the dotted box in Figure 3.2. The audio is encoded and decoded using the new LC3 codec, unless an implementation needs to use a specific, additional codec. Other codecs can be used, but all devices must support some basic LC3 configurations, which are defined in BAP, to ensure interoperability. The encoder produces encoded audio data, which goes into the payloads for the Isochronous Streams.

Once the audio data is encoded, it is placed into SDUs (Service Data Units), which the Core translates into PDUs (Protocol Data Units) which are transmitted using the Isochronous Channels to the receiving device. Once a PDU is received, it is reconstructed as an SDU for delivery to the decoder. The Isochronous Stream is the term used to describe the transport of SDUs, encapsulated in PDUs, from encoder output to the decoder input. It includes retransmissions and any buffering required to synchronize multiple Isochronous Streams. The flow of encoded audio data over an Isochronous Stream is defined in BAP as an Audio Stream, and, like an Audio Channel, is always unidirectional. The relationship of the different terms is illustrated in Figure 3.3.

¹⁴ With the exception of the Presentation Delay, which states when it is rendered. This is explained in Section 3.12.

Section 3.3 - Terminology

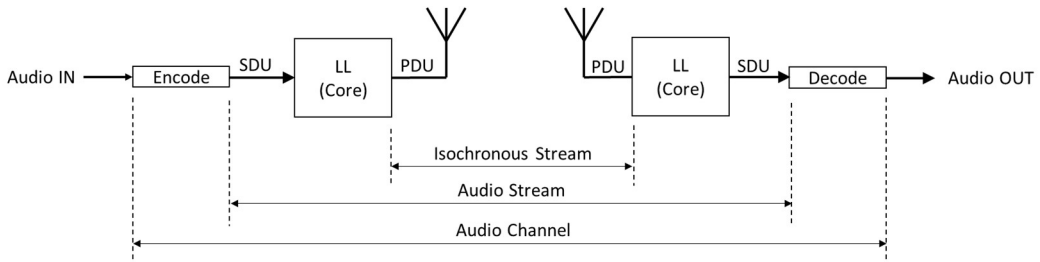


Figure 3.3 Representation of streaming terms

The Core places Isochronous Streams serving the same application together into an Isochronous Group. For unicast streams, it's called a Connected Isochronous Group (CIG), which contains one or more Connected Isochronous Streams (CIS). For Broadcast, it's a Broadcast Isochronous Group (BIG). Where more than one Isochronous Stream is present in an Isochronous Group, carrying audio data in the same direction, the Isochronous Streams are expected to have a time relationship to each other at the application layer. By time relationship, we mean Audio Channels which are expected to be rendered or captured at the same time. A typical application is where the left and right components of a stereo stream are rendered in two separate earbuds. This highlights the fact that a CIG or BIG can contain Isochronous Streams which go to multiple devices.

Bluetooth LE Audio has been designed to be very flexible, which means that sometimes there are different ways of doing things. Taking one example, if we look at a simple connection from a phone to a headset, we need one audio stream going in each direction. One way to do that is to establish separate CISes for each direction.

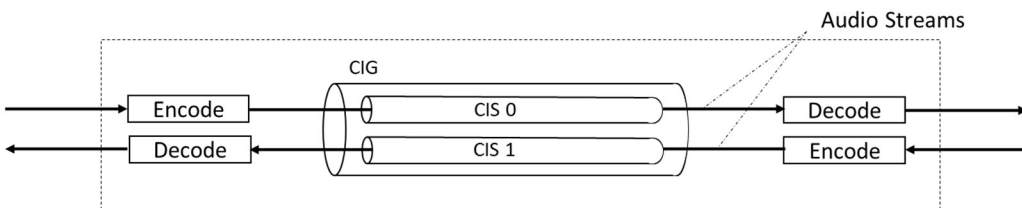


Figure 3.4 Two separate CISes in a CIG

In Figure 3.4 we can see two CISes, an outgoing one (CIS 0) and an incoming one (CIS 1), both contained within the same CIG. We can optimise that by using a single CIS, which is what's shown in Figure 3.5, which shows a single bidirectional CIS (CIS 0), carrying audio data in both directions in the same CIG. We'll see exactly how that works in the next chapter.

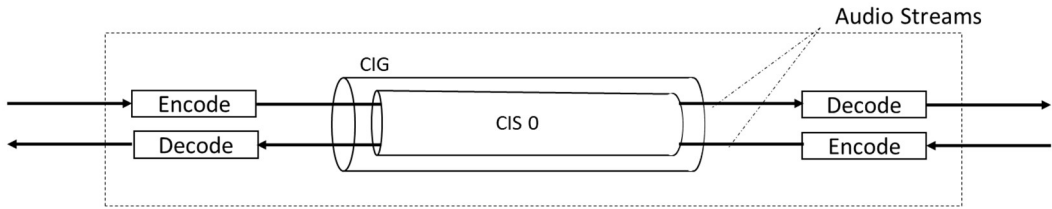


Figure 3.5 A bidirectional CIS in a CIG

Both of these options are allowed; it is up to the application to decide which is the most appropriate for its use. In most cases, the optimization of bidirectionality would be the option of choice, as it saves airtime.

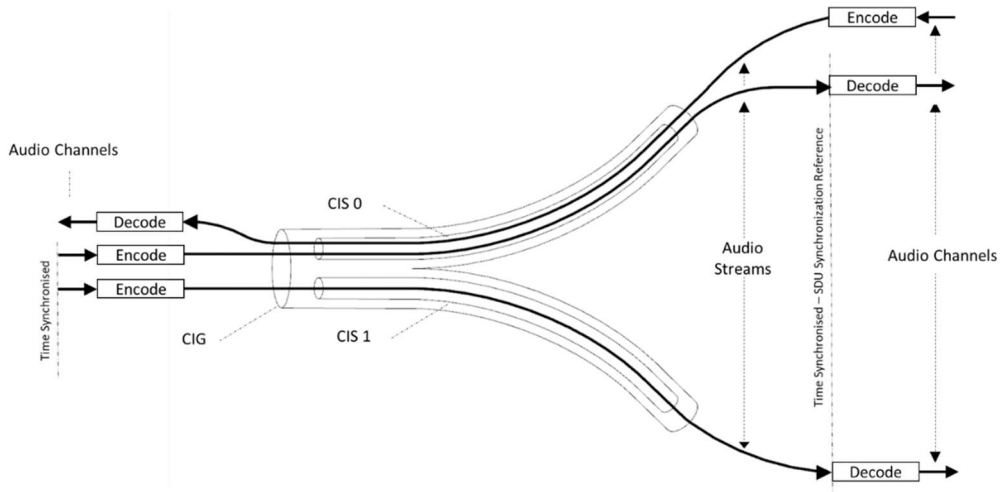


Figure 3.6 A CIG with one unidirectional and one bidirectional CIS serving two Acceptors, e.g., a phone supporting a telephone call with two hearing aids.

Figure 3.6 shows a typical application, with a CIG containing two CISes, which might connect a phone to a pair of earbuds, but where the return microphone is only implemented in one of the earbuds. It shows CIS 0, which is a bidirectional CIS carrying audio from a phone conversation out to the earbud, and audio from the earbud's microphone back to the phone. CIS 1 carries the audio stream from the phone to the other earbud. It is up to the application to determine whether it is a stereo stream that's being sent or mono. In the latter case, the same mono audio data is sent separately over CIS 0 and CIS 1 to the two earbuds. The important point to note here is that a CIG can serve multiple Acceptors, supplying different audio data to each.

3.4 Context Types

To make decisions about which audio streams they want to connect to, devices need to know more about what those streams contain or what they're for. Context Types have been introduced to describe the current use case or use cases associated with an audio stream. Their values are defined in the Bluetooth Assigned Numbers document for Generic Audio. Context

Section 3.4 - Context Types

Types can be used by both Initiators and Acceptors to indicate what type of activity or connection they want to participate in, and are applicable to both unicast and broadcast.

With Bluetooth Classic Audio profiles, the conversation between a Central and Peripheral device is basically “I want to make an audio connection”, with no more information about what it is. As the HFP and A2DP profiles are essentially single purpose profiles, that’s not a problem, but in Bluetooth LE Audio, where the audio stream could be used for a ringtone, voice recognition, playing music, providing satnav instructions, or a host of other applications, it’s useful to know a little more about the intended reason for requesting a stream. That’s where Context Types come in.

As we’ll see later, Context Types are used as optional metadata during the unicast stream configuration process. An Acceptor can expose which Context Types it is prepared to accept at any point in time. For example, if a hearing aid wearer is having a personal (non-Bluetooth) conversation which they don’t want interrupted by a phone call, they can set their hearing aids to be unavailable for a stream associated with a «Ringtone» Context Type. That means that the hearing aids will silently reject an incoming call. That’s more universal than putting their phone on silent, as by using Context Types they will reject an incoming call from any phone they have connected, or a VoIP call from any other connected device. They could also set the «Ringtone» Context Type while they are in a call, to prevent any other call interrupting the current one. This can be done on a per-device basis, meaning you could allow incoming calls to one specific phone, whilst blocking another, providing a powerful method for an Acceptor to control which devices can request an audio stream.

An Initiator uses the Context Type when it is attempting to establish an audio stream, informing the Acceptor of the associated use case. If that Context Type is set to be unavailable on the Acceptor, the configuration and stream establishment process is terminated. This happens on the ACL link before an Isochronous Stream is set up, which means that these decisions can take place in parallel to an existing audio stream without disturbing it, regardless of whether the request is from the device currently providing the stream, or another Initiator wanting to establish or replace an existing stream. It doesn’t matter whether an existing audio stream is unicast or broadcast, which means that a hearing aid user listening to their TV using a broadcast Audio Stream can use Context Types to prevent their current stream being disturbed by any phone call.

There are currently twelve Context Types defined in the Generic Audio Assigned Numbers document, which are exposed in a two-octet bitfield of Context Types, as shown in Table 3.1, with each bit representing a Context Type. This allows multiple values to be used at any one time.

Bit	Context Type	Description
0	Unspecified	Any type of audio use case which is not explicitly supported by another Context Type on a device.
1	Conversational	Conversation between humans, typically voice calls, which can be of any form, e.g., landline, cellular, VoIP, PTT, etc.
2	Media	Audio content. Typically, this is one way, such as radio, TV or music playback. It is the same type of content as is handled by A2DP.
3	Game	Audio associated with gaming, which may be a mix of sound effects, music, and conversation, normally with low latency demands.
4	Instructional	Instructional information, such as satnav directions, announcements or user guidance, which often have a higher priority than other use cases
5	Voice assistants	Man-machine communication and voice recognition, other than what is covered by instructional. It is implied that this is in the form of speech.
6	Live	Live audio, where both the Bluetooth Audio Stream and the ambient sound is likely to be perceived at the same time, implying latency constraints.
7	Sound effects	Sounds such as keyboard clicks, touch feedback and other application specific sounds.
8	Notifications	Attention seeking sounds, such as announcing the arrival of a message.
9	Ringtone	Notification of an incoming call in the form of an inband audio stream. The Ringtone Context Type is not applied to an out of band ringtone, which is signalled using CCP and TBS.
10	Alerts	Machine generated notifications of events. These may range from critical battery alerts, doorbells, stopwatch and clock alarms to an alert about the completion of a cycle from a kitchen appliance or white goods.
11	Emergency alarm	A high priority alarm, such as a smoke or fire alarm.
12 - 15	RFU	Not yet allocated.

Table 3.1 Currently defined Context Types

Section 3.4 - Context Types

Most of these are obvious, but two of the Context Types are worth special attention: «Ringtone» and «Unspecified»¹⁵.

The «Ringtone» Context Type is used to announce an incoming phone call, but only where there is an inband¹⁶ ringtone which requires an audio stream to be set up. If the user was already receiving an audio stream from a different device to the one with the incoming phone call, this could be problematic. To signal the incoming call to the user without dropping the current audio stream would require at least one of the earbuds to set up a separate unicast stream from the second Initiator. In practice, many Acceptors are unlikely to have the resources to support concurrent streams from different Initiators at the same time. The Acceptor could drop the current audio stream to switch to the inband ringtone, but if they then reject the call, they'd need to restore the original Audio Stream. In most cases, a better user experience is likely to result from providing an out of band ring tone, which is generated by the earbuds using CCP and TBS. The user can hear this mixed into their existing audio output and decide whether to accept or reject the call. If they reject the call, they can continue listening to their original stream. In most cases, «Ringtone» is best used to manage whether devices are allowed to interrupt the current audio application with incoming phone calls. We will cover how out of band ringtones are handled in Chapter 9.

The «Unspecified» Context Type is a catch-all category. Every Acceptor has to support the «Unspecified» Context Type, but doesn't need to make it available. When an Acceptor does set the «Unspecified» Context Type as available, it is saying that it will accept any Context Type other than ones it has specifically said it will not support. As implementers get used to Context Types, some will use this to allow the Central device (typically a phone) to be in charge of the audio use case in much the same way as it is for A2DP and HFP. An Acceptor that just sets «Unspecified» as available is effectively allowing the phone to take full control of what it can be sent. We'll discuss the concepts of Supported and Available in more detail in Chapter 7.

All of the Context Types are independent of each other. The use of any one of them does not imply or require that another one needs to be supported, unless that requirement is imposed by a top level profile. As an example, support for the «Ringtone» Context Type does not generally imply or require support for the «Conversational» Context Type, as «Ringtone» could be used by itself for an extension bell to alert someone with hearing loss of an incoming phone call on a landline phone.

Context Types are used by both Initiators and Acceptors to provide information about the

¹⁵ When Context Types are used in text, the convention is to distinguish them from surrounding text by enclosing them in guillemets, i.e. « and ».

¹⁶ An inband ringtone is one where the sound is carried in an Audio Stream from the phone, so you can hear your customised ringtone or message. In contrast, an out of band ringtone is a sound generated locally in the Acceptor, so only needs a control signal, not the presence of an Audio Stream.

use case intended for a stream, allowing them to make a decision about whether to accept a stream. To accomplish this, they are used in a range of characteristics and LTV¹⁷ metadata structures. You'll find them used in this way in the following places:

- Supported_Audio_Contexts characteristic [PACS 3.6]
- Available_Audio_Contexts characteristic [PACS 3.5] (also used in a Server announcement – [BAP 3.5.3])
- Preferred Audio Contexts LTV structure (metadata in a PAC record) (see also [BAP 4.3.3])
- Streaming Audio Contexts LTV structure (metadata used by an Initiator to label an audio stream) [BAP 5.6.3, 5.6.4, 3.7.2.2]

An Audio Stream can be associated with more than one Context Type, although the intention is that the Context Type value represents the current use case. The Streaming Audio Contexts metadata has procedures to allow a device to update the bitfield values as the use case changes. This is typically used where an established stream is used for multiple use cases. An example would be where an Audio Source mixes in audio from different applications, such as a satnav message that could interrupt music. In this case, it is efficient to continue to use the same stream, assuming its QoS parameters are suitable, and just update the current Context Type value for the duration of the new use case.

3.5 The unconnected model

With classic Bluetooth, a common connection model has evolved for audio applications, where the Central device makes a connection and then maintains it for as long as possible, regardless of whether it's using it. The advantage of this approach is that when an event happens, such as an incoming call arriving, the connection to the earbuds already exists, so the ringtone in the earbuds can appear almost immediately. The origin of this behaviour dates back to the early days of Bluetooth, when it was assumed that most people would use it primarily for their phone and headsets, so that the connection between them could be persistent. A lot of work has gone into reducing the power required to keep the link alive, resulting in sniff mode and the more recent sniff sub-rating. But both apply to a persistent connection. Whilst it solves one problem, which is the latency of a new connection, it leads to several others, which have been persistent issues with the Bluetooth audio experience.

The first of these is that it assumes that a user has a single phone and set of earbuds. As soon as they get a second headset, the Central needs to resort to “device picker” or “pick list” applications to change what they are using. When a call comes in on a different device, such as a laptop or second phone, the first Central can be reluctant to relinquish control, leading to

¹⁷ LTVs are triplet structures, consisting of a statement of the Length of the structure, its Type and the parameter Values in that order.

Section 3.6 - Availability

some unexpected behaviour. Adding in different applications, such as switching between music on one device to telephony on another one leads to the well-known multi-profile issues. A lot of work has been put into making the most common of these use cases work, but it is not easily scalable as consumers want to use more and more Bluetooth devices for increasingly complex use cases.

One of Bluetooth LE Audio's goals was to try to get rid of multi-profile issues by design, which requires a number of fundamental changes in the connection mode. The first change we've just seen, with Context Types providing information about the use case of the Audio Stream. The second one is a fundamental change to the connection model, moving away from the "always connected" model of classic, to a new unconnected model, where devices can announce their availability to take part in a use case. This allows far more flexibility. They may still keep ACL connections, but can drop their isochronous stream when they don't need them, in the knowledge that they can establish new ones quickly with the same, or other devices, whenever they are needed.

3.6 Availability

To enable devices to make informed choices about what they're doing, they not only need the ability to tell each other about the use case they're engaging in (which is the reason for the Context Types), but also which future ones they are interested in participating in, as the current use case changes. This is where Availability comes in.

As we've seen above, Context Types are used by Acceptors to signal whether they can take part in a use case. That's accomplished in two ways. An Acceptor uses the `Available_Audio_Contexts` characteristic defined in PACS to state which of its Supported Audio Context Types can currently be used to establish an Audio Stream. Audio Sources, whether Initiators or Acceptors, also use the `Streaming_Audio_Contexts` LTV structure in the metadata of their codec configurations, to inform an Audio Sink of the use case(s) that are associated with an Audio Stream.

Bluetooth LE Audio devices wanting to establish unicast Audio Streams can also use Context Types to signal their availability before they even start an Audio Stream by including the `Streaming_Audio_Contexts` LTV structure in their advertising PDUs. In a similar way, Initiators, acting as Broadcast Sources, include this in the metadata section of their Periodic Advertisements, so that Broadcast Sinks and Broadcast Assistants can filter out the use cases they want from those they are not interested in receiving.

For Broadcast Sources, you should note that if the `Streaming_Audio_Contexts` field is not present in a Codec ID's metadata (which we'll get to in Chapter 4), it will be interpreted as meaning that the only `Supported_Audio_Contexts` value is «Unspecified», so every Broadcast Sink can synchronize to it, unless they have specifically set «Unspecified» as non-available.

3.7 Announcements

The new unconnected environment means that new processes have been introduced to allow the rapid establishment of connections. A key part of this is the introduction of Announcements. Announcements are used by both Initiators and Acceptors in both unicast and broadcast topologies as a means of conveying information about a use case and a device's readiness to participate in a use case.

Announcements also support the philosophy of giving more autonomy to an Acceptor that is involved in a unicast connection. The Bluetooth LE Audio Specifications allow Acceptors to transmit Announcements, informing Initiators that they are available to receive or transmit audio data, by using a Service Data AD Type¹⁸. An Acceptor can decide whether to use a Targeted Announcement, where it is connectable and requesting a connection, or a General Announcement, where it is simply saying that it is available, but not initiating a specific connection. Announcements can be received and acted on by multiple Acceptors.

3.7.1 Unicast Announcements

For unicast use cases, an Acceptor can send an Announcement in an advertisement, using an LTV structure used in the AD Type field of its ASCS UUID as shown in Table 3.2.

Field		Size (Octets)	Description
Length		1	Length of Type and Value fields
Type		1	Service Data UUID (16 bit)
Value	ASCS UUID ¹⁹	2	0x0184E
	Announcement Type	1	0x00 = General Announcement 0x01 = Targeted Announcement
	Available_Audio_Contexts	4	Available_Audio_Contexts characteristic (from ASCS)
	Metadata Length	1	≥ 1 if there is additional metadata, otherwise 0
	Metadata	varies	Metadata in LTV format

Table 3.2 AD Values for Targeted and General Announcements

Announcements can be either General or Targeted, depending on the Acceptor's requirements for the use case. There is a further subtlety in Announcements, which is that they may or may

¹⁸ AD Data Types are structures used in Bluetooth advertisements to provide information about a device or its capabilities. They are defined in the Core Specification Supplement (CSS).

¹⁹ A UUID is a universally unique identifier, defined in the Bluetooth 16-bit UUIDs Assigned Numbers document, and used to identify a particular service or characteristic.

Section 3.7 - Announcements

not include a Context Type value, which depends on whether they are used in a BAP or a CAP context. We'll explore the details of that in Chapter 6. For the moment, the important thing to understand is that Announcements allow Acceptors to stay in an unconnected state when they have no audio data to send or receive, but gives them a means to make a rapid connection once that situation changes.

3.7.2 Broadcast Announcements

Broadcast Sources use Announcements within their Extended Advertisements to inform Broadcast Receivers and Broadcast Assistants (see Section 3.13) that they have a broadcast stream available. They are an essential feature for broadcast scanners to determine the existence and content of broadcast streams. Two of these Broadcast Announcements are defined in BAP, and a third in PBP.

3.7.2.1 Broadcast Audio Announcements

Broadcast Audio Announcements are defined in BAP and inform any scanning device that a Broadcast Source is transmitting a group of one or more broadcast Audio Streams. The LTV structure used for Broadcast Audio Announcements is shown below in Table 3.3.

Field		Size (Octets)	Description
Length		1	Length of Type and Value fields
Type		1	Service Data UUID (16 bit)
Value	Broadcast Audio Announcement Service UUID	2	0x01852
	Broadcast_ID	3	A random ID, fixed for the life of the Broadcast Isochronous Group
	Supplementary Announcement Service UUIDs	varies	Optional UUIDs defined by top level profiles

Table 3.3 AD Values for Targeted and General Announcements

3.7.2.2 Basic Audio Announcement

The confusingly similarly titled Basic Audio Announcement (also defined in BAP) is used in the Periodic Advertising train by a Broadcast Source to expose the broadcast Audio Stream parameters through the Broadcast Audio Stream Endpoint structure (BASE). Its use is described in Chapter 8.

3.7.2.3 Public Broadcast Announcement

Public Broadcast Announcements were introduced in PBP to provide more information for devices scanning for the presence of broadcasts. Most of the detailed information about broadcasts is included in Periodic advertisements, which are the last of the advertisements that a scanner will access. The Public Broadcast Announcement has been designed to provide key

information about a broadcast in Extended Advertisements, effectively acting as a filter to limit a scanner's need to read every Periodic Advertising train. The information included in a Public Broadcast Announcement is that which allows a scanner to determine whether a broadcast stream complies with the requirements for an Auracast™ transmission, and whether or not it is encrypted. The presence of a Public Broadcast Announcement can significantly reduce the scanning requirements for a Broadcast Receiver or a Commander (See Section 3.13). We'll look at exactly what the Public Broadcast Profile does in Chapter 11.

3.8 Content Control ID - CCID

A consequence of separating the control and data planes in Bluetooth LE Audio is that there is no longer any direct relationship between a control signal related to the current use case, such as phone control or media control, and the audio stream. That adds flexibility to Bluetooth LE Audio, but results in a little more complexity. An Initiator might have multiple applications that are running concurrently, where a user may want to associate control with a specific one of those applications. Think of the case of two separate telephony calls, such as a cellular call and a concurrent Teams meeting, where the user may want to put one call on hold, or terminate one whilst retaining the other. To cope with this situation, the Content Control ID has been introduced, which associates a Content Control service instance with a specific unicast or broadcast stream.

The statement that CCIDs can be used for broadcast might seem contradictory, but highlights the fact that although broadcast streams don't need an ACL connection, there are many applications where one might be present. For example, if you transition from using unicast to listen to a music stream on your phone to broadcast, so that you can share it with your friends, you would still expect to be able to control the media player. In this case you would keep the ACL link alive and associate the media controls with the broadcast stream.

The Content Control ID characteristic is defined in Section 3.45 of the GATT Specification Supplement²⁰ as having a value that uniquely identifies an instance of a service that either controls or provides status information on an audio-related feature. It is a single octet integer which provides a unique identifier across all instances of Content Control services on a device.

When an Audio Stream contains content which is controlled by a content control service, it includes the CCID in a list of such services in the Audio Stream's metadata to tell both Acceptors and Commanders where they can find the correct service. We'll look at Commanders in a moment in Section 3.13.

CCIDs are currently only used for the Telephone Bearer Service and the Media Control Service. They do not apply to rendering or capture control.

²⁰ This is a document that describes generic features used within Bluetooth LE.

3.9 Coordinated Sets

Despite the fact that we've only had them for a few years, we're already so familiar with the concept of TWS earbuds, that most people forget that the Bluetooth Classic Audio specifications don't cover the way they work. As explained in Chapter 1, they all rely on proprietary extensions from silicon chip companies. The design of Isochronous Channels rectifies that, allowing an Initiator to send separate Audio Streams to multiple Acceptors, along with a common reference point at which all Acceptors know they have received the audio data and can start decoding it. This means that an Initiator needs to know that pairs of Acceptors, which are separate devices, can be treated as a single entity.

The concept of a Coordinated Set addresses that need. It allows devices to expose the fact that they are part of a group of devices which together support a common use case and should be acted on as an entity. Whilst most people will immediately think of a pair of earbuds or hearing aids, that set could equally be a pair of speakers or a set of surround-sound speakers. A Coordinated Set of earbuds should be represented as a single device, so that anything that happens to one, happens to the other, although how that happens may be down to the implementation. When you adjust the volume for a pair of earbuds, the volume of both should change at the same time²¹, and if you decide to listen to a different Audio Source, that change should occur simultaneously on both left and right earbuds. You do not want a user experience where your left earbud is listening to your TV, whilst the right earbud is streaming music from your phone.

Coordination is handled by the Coordinated Set Identification Profile and Service (CSIP and CSIS²²), which are referenced from within CAP. The main feature of CSIS and CSIP, aside from identifying devices as members of a Coordinated Set, is to provide a Lock function. This ensures that when an Initiator interacts with one of the members, the others can be locked, preventing any other Initiator from interacting with other members of that set.

The need for such a Lock is that ear-worn devices, such as hearing aids and earbuds, have a problem with communicating directly with each other using Bluetooth technology. That's because the human head is remarkably good at attenuating 2.4GHz signals. If an earbud is small, which means that its antenna will also be small, it is unlikely that a transmission from a device in one ear would be received by its partner in the other ear. Many current earbuds get around this limitation by including a different, lower frequency radio within the earbud for ear-to-ear communication, using a technology which is not significantly attenuated by the head, such as Near Field Magnetic Induction (NFMI). This second radio adds cost and takes up space, but removing it and relying on the 2.4GHz Bluetooth link between the earbuds raises

²¹ The Volume Control profile has the flexibility to allow these to be changed independently if the user prefers.

²² Taken together, CSIP and CSIS are often referred to as CSIPS.

the risk that different Initiators could send conflicting commands to left and right earbuds.

With CSIP and CSIS, if you accept a phone call on your left earbud, the Initiator would set the Lock on both earbuds, and transition your right earbud to the same stream before releasing the Lock. The Lock feature allows these interactions to be managed without the need for the members of the Coordinated Set to talk to each other.

If members of a Coordinated Set do have another radio connection which can penetrate the head, the Hearing Access Service has a feature which will signal that this radio can be used to convey information to the other hearing aid. At the moment this feature is limited to information on preset settings, but may be used for other features in the future.

Generally, members of a Coordinated Set are configured at manufacture and shipped as a pair, but membership can be set to be written as well as read, allowing for later configuration, or the replacement of faulty or lost units.

3.10 Audio Location

With every previous Bluetooth audio specification there was a single Bluetooth LE Audio source and a single Bluetooth LE Audio sink. The audio stream was sent as mono or stereo and it was up to the audio sink to interpret how it was rendered. A stereo stream would typically be encoded as joint stereo, and after decoding it, an earbud, hearing aid or speaker would discard the audio information it didn't need. This has the advantage of simplicity, as only one stream is ever transmitted, but it results in the transmission of redundant information.

Bluetooth LE Audio is intended to address applications with multiple speakers or earbuds, whilst supporting multiple different audio streams, which may have different qualities or different languages. It has the ability to optimise airtime for each Audio Sink, by only sending a device the audio stream it needs. In general, for a pair of earbuds, the left earbud only receives the left audio stream, and the right earbud only receives the right audio stream. That means that each Audio Sink can minimise the time that its receiver is on, thereby reducing its power consumption. It also improves robustness, as the smaller packets are less susceptible to interference.

To accomplish this, devices need to be configured to know what spatial information they are meant to receive, for example, a left or a right stream from a stereo input. They do this by specifying an Audio Location, which is defined in BAP as the “intended logical spatial rendering location of an Audio Channel within a spatial arrangement of loudspeakers or other audio transducers that render audio”. In effect, it is a statement of what the device is, for example a left earbud, or a right speaker.

The concept of an Audio location has proven to be quite confusing. In most cases, a device configured as being Front Right would always expect to receive the right stream of a stereo signal. For unicast applications, that will happen because the earbud will have informed the

Section 3.10 - Audio Location

Initiator that it has an Audio Location value of Front Right. It is up to the application in the Initiator to ensure that it sends the correct audio data to each device. However, that can be changed by the Initiator's application, which can direct whatever audio stream it wants to each CIS. That may sound counter-intuitive, but there may be applications where it makes sense. Think of people sitting in a tennis court listening to a commentary on the match. If people on both sides of the tennis court received the same stereo input, those on one side of the court would hear the ball going the wrong way. In this case, they might want to "mirror" the sound, swapping the left and right audio content. This does not affect their Audio Location – it would be effected by the Initiator swapping the content for each CIS.

For broadcast, an earbud or speaker would use its Audio Location to determine which audio stream it wanted to receive, with each earbud independently making that choice. In most cases, the earbuds would also share their Audio Locations with an associated remote control device, which could perform the same mirroring operation by reconfiguring the Audio Location values of each earbud.

With unicast streams, it is up to the Initiator to decide what content is sent to each Audio Location, so the application is in control of what is received. This will be explained when we look at how to configure a CIS in Section 7.4 of Chapter 7.

With broadcast stream, where there is no link between the Broadcast Transmitter and the earbuds, a slightly different situation presents itself. A Broadcast Transmitter has no idea of what devices are listening to it. To accommodate that, many will transmit three different audio streams – a left stream, a right stream and a mono stream. In this case, an Acceptor which can accept a left stream needs to decide whether to synchronize to the left stream or the mono stream. There should be a default, which a user can configure, or an option to select this from a user interface, which may be on a remote control device.

Audio Locations are defined in the Bluetooth Generic Audio Assigned Numbers and follow the categorization of CTA-861-G's Table 34 codes²³ for speaker placement. They are stored in Sink and Source Audio Location characteristics, which are defined in PACS. The values are expressed as bits in a four-octet wide bitfield. The most common Audio Locations are shown in Table 3.1.

²³ https://archive.org/stream/CTA-861-G/CTA-861-G_djvu.txt

Audio Location	Value (bitmap)
No specified Audio Location (e.g. mono audio)	0x00000000
Front Left	0x00000001 (bit 1)
Front Right	0x00000002 (bit 2)
Front Centre	0x00000004 (bit 3)
Low Frequency Effects 1 (Front Woofer)	0x00000008 (bit 4)
Back Left	0x00000010 (bit 5)
Back Right	0x00000020 (bit 6)

Table 3.4 Common Audio Location values

Every Acceptor which can receive an audio stream must set at least one Audio Location. An Audio Location is normally set at manufacture, but in some cases it may be changeable by the user – for instance, a speaker could have an application or a physical switch to set it to Front Left, Front Right or Unassigned, which is interpreted as Mono.

In the first release of the Bluetooth LE Audio specifications, mono was not defined. This followed the concepts of the CTA-861 specifications for digital TV, which considered that mono is not a location, but a property of the stream. That caused massive confusion among developers, as well as introducing some implementation issues, as a Broadcast Receiver could only indicate that it wanted to select mono by omitting its Audio Location characteristic. A recent update to the specification now requires a mono device to set its Audio Location value to 0x00000000 if it wants to receive a mono stream.

Currently, there is no definition of Audio Location for an Audio Source, such as a microphone, but the assumption is that the Audio Location refers to where the audio has been captured. In most cases, captured audio is a single mono stream, so has no Audio Location, although pair of earbuds may want to differentiate between right and left if both microphones are being sampled and transmitted to help with noise cancellation.

We'll look further at exactly how Audio Locations are used in Chapter 5, but now we've introduced them, we can move on to look at Channel Allocation.

3.11 Channel Allocation (multiplexing)

You always knew what was being transported in the Bluetooth Classic Audio profiles. HFP carries a mono audio stream. A2DP has more options, as the mandatory SBC codec can encode streams as mono, dual channel, stereo or joint stereo. In most implementations joint-stereo is used, so all devices receive both left and right channels. In contrast, LC3 encodes a single channel. To accommodate this, Bluetooth LE Audio is a lot more flexible, allowing a CIS or BIS to contain one or more audio channels multiplexed into a single packet, limited only by the available bandwidth.

Section 3.11 - Channel Allocation (multiplexing)

The reason for this approach is that the LC3, which is the mandatory codec for all Bluetooth LE Audio implementations is a single channel codec. This means that it encodes each audio channel separately into discrete frames of a fixed length. With SBC, joint stereo coding, which combines both left and right audio channels into a single encoded stream was popular because it was more efficient than encoding two separate left and right channels. In the thirty years since SBC was written, codec technology has moved forward. The more efficient design of LC3 means that there is very little advantage in joint stereo coding compared to encoding each channel separately. This allows radio on-time and power consumption to be optimised, by only transmitting the audio data that each device needs, rather than making everything receive both stereo channels and then throw unwanted data away.

However, some implementations may still want to concatenate multiple Bluetooth LE Audio channels into a single packet. This will be slightly more efficient for devices like soundbars and stereo headsets, where only a single isochronous channel is needed. In these applications, Channel Allocation [BAP Section 4.2] can be used to concatenate the individual encoded frames, allowing multiple Audio Channels to be grouped together.

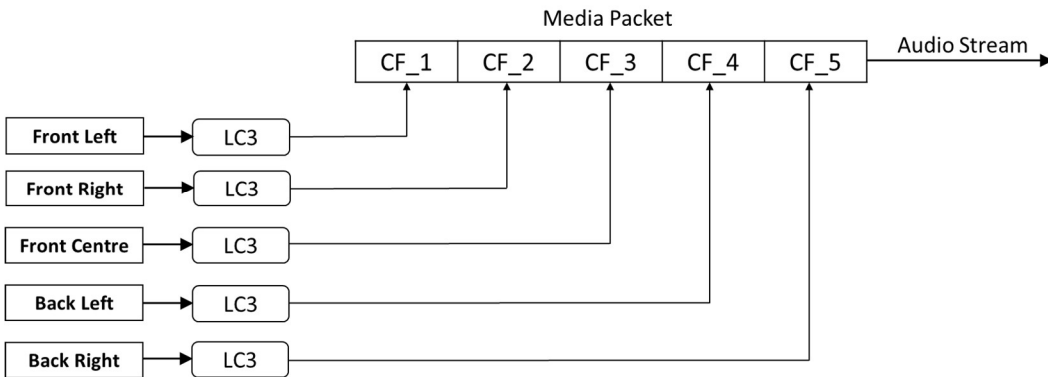


Figure 3.7 Example of multiplexing multiple Bluetooth LE Audio Channels

Figure 3.7 shows an example of how this works for a five-channel surround-sound system. Five audio input channels are separately encoded using LC3 and the encoded frames are then arranged into a Media Packet which is transmitted as a single isochronous PDU. The LC3 codec frames are always arranged in ascending order of the Published Audio Capability Audio Location associated with each audio channel, using the Assigned Numbers which were summarised in Table 3.4. So, in this case, they will be ordered as Front Left (0x0000000001), Front Right (0x0000000002), Front Centre (0x0000000004), Back Left (0x0000000010) and Back Right (0x0000000020). Note that mono cannot be included in a Media Packet, as the Audio Location value of 0x00000000 cannot be indicated at the same time as any other Audio Location bit.

The Media Packet contains only encoded audio frames. It can be expanded to include multiple blocks of encoded audio frames, each of which include one frame for each of the Audio Locations, as shown in Figure 3.8. CF_N₁ refers to frames from the first sample of each of

the incoming audio channels; CF_N₂ to those from the next samplings of those audio channels.

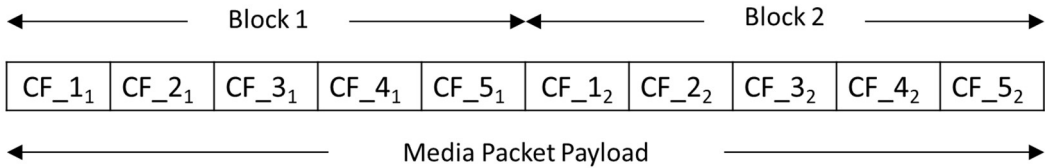


Figure 3.8 Media Packet containing two blocks of five Audio Channels

Using blocks may look efficient, but it comes with some important caveats. It results in larger packets, which are more vulnerable to interference. It also increases the latency, as the Controller needs to wait for multiple frames to be sampled before it can start transmission. If multiple blocks are added to the multiple Isochronous Channel features of Burst Number or Pre-Transmission Offset, which we'll come to in the next chapter, it very quickly results in latencies that can be hundreds of milliseconds. There are some occasions where that may be useful, but not in the general audio applications we use today.

There is no header information associated with the media packet. Instead, the number of Audio Channels that can be supported are specified in the Supported_Audio_Channel_Counts LTV which is included in the Codec Specific Capabilities LTV in the PAC records, with the value for each Isochronous Stream being set by the Initiator during the stream configuration process. The number of blocks being used is configured at the same time using the Codec_Frame_Blocks_Per_SDU LTV structure.

We'll look at these in more detail in Chapter 5 when we discuss the LC3 and Quality of Service.

3.12 Presentation Delay and serialisation of audio data

Supporting two earbuds brings us to another issue that Bluetooth LE Audio had to solve, which is ensuring that the sound at both the left and the right ear is rendered at exactly the same time. In the past, audio data was sent to a single device, which knew how to extract the left and right signals and render them at the same time. With Bluetooth LE Audio, CISes carry data to different destinations serially, using different transmission slots. Although the incoming Audio Channels present data to the Initiator at the same point in time, the encoded packets arrive at the Acceptors one after the other. They may be further delayed by retransmissions. Figure 3.9 illustrates this by adding examples of left and right packets going to two acceptors onto the CIG diagram of Figure 3.6

Section 3.12 - Presentation Delay and serialisation of audio data

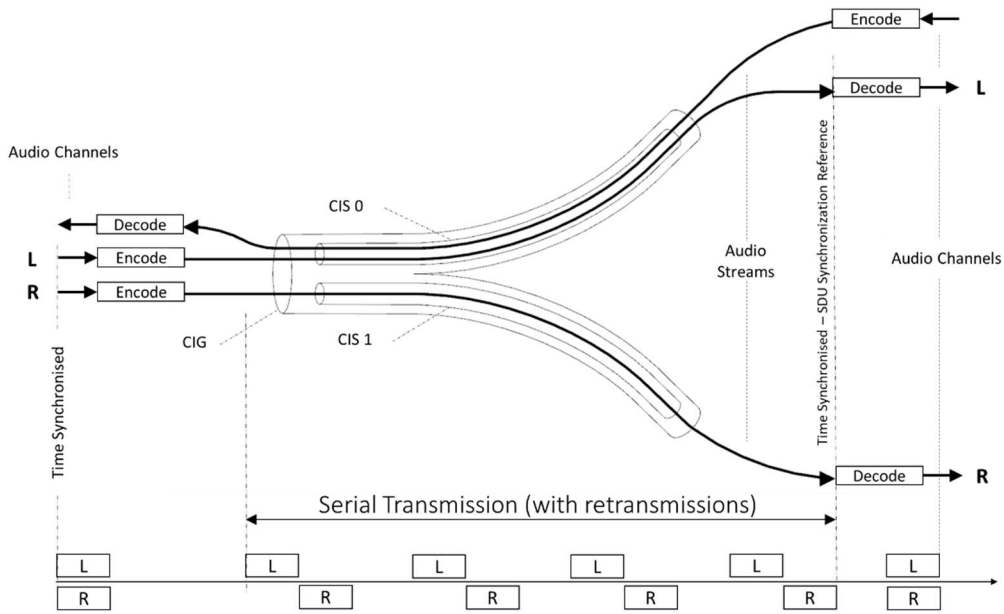


Figure 3.9 The serial transmission of audio data

This serialisation causes a problem. The human head is remarkably good at detecting a difference in the arrival time of sound between your left and right ears, and uses that difference to estimate the direction from which the sound is coming.

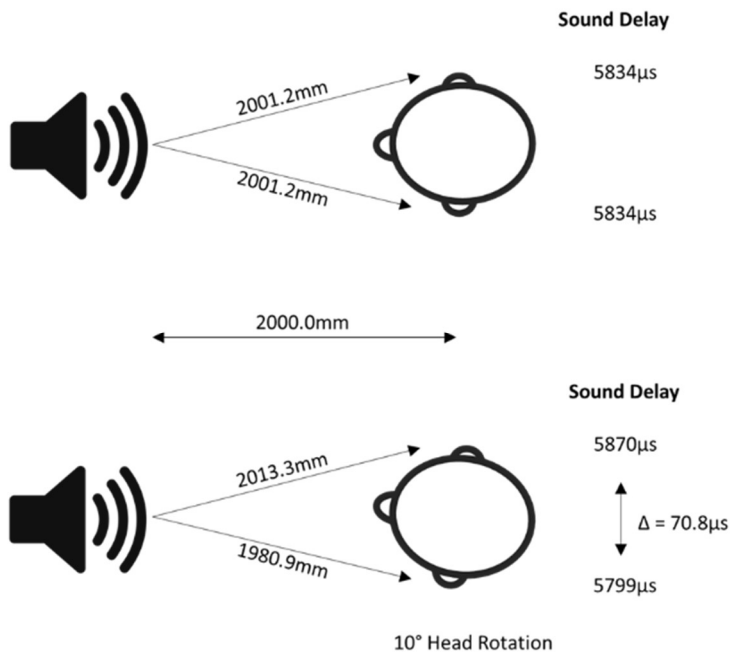


Figure 3.10 Effect of head rotation on sound arrival

Figure 3.10 illustrates that if you're two metres away from an audio source, and you rotate your head by just 10 degrees, that equates to just over a 70 μ s difference in the arrival time of the sound. If there's a variation in the rendering time between your left and right ear, your brain interprets this as the sound source moving. If that difference is much more than 25 microseconds and changes regularly, you start to get an unpleasant effect, where it feels as if the sound is moving around within your head. To prevent that, it's important to have a synchronization technique to ensure that the left and right earbuds always render their respective audio data at exactly the same time.

We can't rely on any Bluetooth communication between two earbuds. As we've said before, the human head is very efficient at attenuating a 2.4GHz signal, as it contains a lot of water. If you have small earbuds, which fit neatly in the ear canal, there is no guarantee that they will be able to communicate with each other.

There are two parts to the Bluetooth LE Audio solution. First, both earbuds need to know a common synchronization point, which is the point in time at which every Acceptor can guarantee that every other Acceptor has had every possible chance to receive a transmitted packet, whether it's a unicast or a broadcast stream. This point in time has to be provided by the Initiator, as it is the only device which knows how many attempts it will take to send packets to all of the Acceptors. (Remember that the Acceptors generally can't talk to each other and may well be unaware of each other's existence.)

In most cases, the Acceptors will have received their audio data packets earlier than that common synchronization point, as in audio applications data packets are scheduled to be retransmitted multiple times to maximise the chance of reception. That is because of the problem of dropouts in an audio stream, as a result of missing packets. These are particularly annoying artefacts for the listener, so retransmissions are used to help improve the robustness of the signal. This means that every Acceptor needs to include enough buffering to store packets from the earliest possible arrival time – the time when their packet is first transmitted, until the common Synchronization Point. We'll learn more about the Synchronization Point in Chapter 4.

However, you can't render the audio at the Synchronization Point, as it's still encoded. Between the Synchronization Point and the final rendering point the data needs to be decoded, and any additional audio processing, such as Packet Loss Concealment²⁴ (PLC), active noise cancellation (ANC), or hearing aid audio adjustments performed, before it can finally be rendered.

²⁴ Packet Loss Concealment is a technique that attempts to recreate missing or corrupted packets of audio data, generally based on what the previous packets contained. It is an attempt to avoid audible artefacts when an audio stream is disrupted.

Section 3.12 - Presentation Delay and serialisation of audio data

The time needed for this processing may vary between different Acceptors. Whilst you expect a pair of hearing aids, speakers or earbuds supplied as a pair from one manufacturer to be designed to have the same processing time for each earbud, it could be different if the Acceptors come from different manufacturers, or even if a firmware update is applied to one, but not the other. For this reason, the Bluetooth LE Audio specification includes the concept of Presentation Delay. Presentation Delay is a parameter whose value is set by an Initiator, specifying the time in microseconds after the Synchronization Point when the audio is to be rendered in every Acceptor. This is illustrated in Figure 3.11. The “C>P” suffix for the SDU Synchronization Point refers to the Central to Peripheral direction (Initiator to Acceptor). LL refers to the Link Layer in the Controller, which is where the data being sent over the air is received.

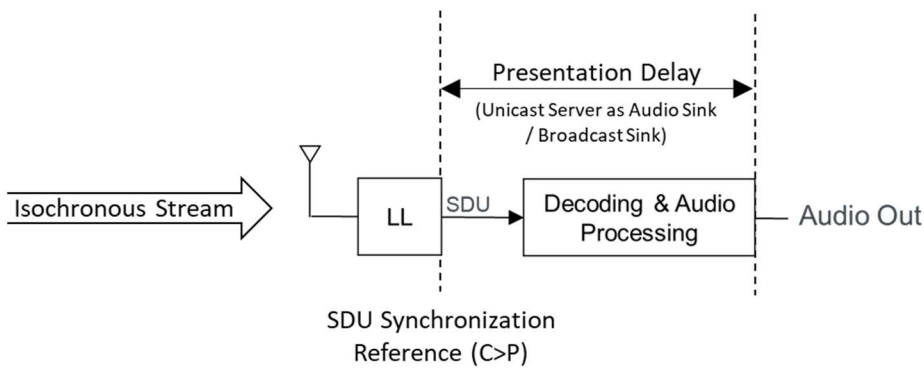


Figure 3.11 Presentation Delay for rendering on an Acceptor

The Presentation Delay for rendering may also include a Host application dependent element to deliberately increase the time until rendering. This is a commonly used technique for audio streams linked to video, where it can be used to delay the rendering point to compensate for lip-synch issues. With public broadcast applications, where there may be multiple Broadcast Transmitters covering a large auditorium or stadium, Presentation Delay may also be used to set specific delays to compensate for differing distances between the Broadcast Transmitters serving each audience group and the audio source. Sound travels 343m every second, so it can take half a second for sound to propagate across a large stadium. For this reason, venues often apply delays to speakers to help synchronize the sound in different sections of the venue. The same effect can be achieved using Presentation Delay with multiple Bluetooth LE Audio Broadcast Transmitters to bring the audience closer to the origin of the sound.

Every Acceptor contains values for the minimum and maximum Presentation Delay it can support. The minimum represents the shortest time in which it can decode the received codec packets and perform any audio processing before it renders the sound; the maximum reflects the longest amount of buffering it can add to that. These are read by an Initiator during configuration, which must respect the maximum and minimum values for all Acceptors within each stream it is transmitting. That means it cannot set a value greater than the lowest Presentation_Delay_Max value of any of the Acceptors, or lower than the highest value of

Presentation_Delay_Min of any of them. Acceptors may also expose their preferred value for Presentation Delay, which an Initiator should attempt to use, unless an application on the Initiator needs a specific value, as it might for live applications or to compensate for lip-synch. It is not expected that Presentation Delay would be exposed to device users receiving the audio, as it is always set by the application on the Initiator

Presentation Delay was designed to provide a common rendering point for multiple Acceptors. It supports the situation where the Acceptors might not be aware of each other's existence, such as a user with hearing loss in one ear, who would wear a hearing aid and a single earbud. As the same Presentation Delay is applied to both, both devices would render at the same time. In most applications, the value for Presentation Delay should be as small as possible. Supporting higher values of Presentation Delay increases the burden on the Acceptor's resources, as it needs to buffer the decoded audio stream for longer.

For Broadcast, when there is no connection between an Initiator and an Acceptor, the Initiator needs to make a judgement of what value of Presentation Delay will be acceptable to all potential Broadcast Sinks, based solely on its application. In general, values above 40ms (which is a value every Acceptor must support) should be avoided, as they may result in echo if the ambient sound is also present²⁵. The specifications do not define what a Broadcast Sink should do if the Presentation Delay falls outside the range it can support.

To allow even lower latencies for broadcast applications, a Broadcast Transmitter can include a special LTV flag in its Broadcast Audio Stream Endpoint, called the Broadcast Audio Immediate Rendering Flag. When this is present, an Acceptor can make a unilateral decision to render the incoming stream as quickly as it can, ignoring the value of Presentation Delay set by the Broadcast Source. Where there is a pair of earbuds or hearing aids, they need to agree on what this value is, but how they do that is outside the specification. It would normally be set during manufacture.

For both unicast and broadcast, a top level profile may specify a specific value for Presentation Delay, particularly if they are supporting low latency applications. For example, where an Audio Stream also has ambient sound present, they may require that the «Live» Context Type is used, along with a Presentation Delay setting of not more than 20ms (for HAP or TMAP support).

Presentation Delay is also applied to audio capture, where audio data is travelling from an Acceptor to an Initiator (denoted in the specifications as P>C, or Peripheral to Central). Here, it represents the time from the point that audio is captured, then subsequently processed, sampled and encoded, to the reference point where the first packet of the first Isochronous

²⁵ There are valid reasons for using them in large venues, where they can be adjusted to ensure synchronisation with an ambient source, but this is a fairly specialised application.

Section 3.12 - Presentation Delay and serialisation of audio data

Stream could be transmitted, as shown in Figure 3.12.

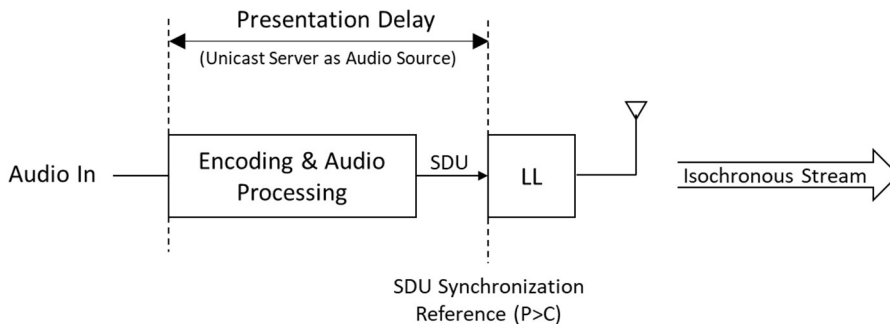


Figure 3.12 Presentation Delay for audio capture

Using Presentation Delay in audio capture ensures that every microphone or other audio transducer captures the sound at exactly the same point in time. It defines an interval during which every Acceptor can prepare its audio packets for transmission, with the first transmission occurring at the end of the Presentation Delay. All other audio data sources will then transmit their encoded packets in their allocated transmission slots. This is desirable when a phone wants to combine the microphone data from left and right earbuds, as it can use the knowledge of the common capture times to combine them. Rather than simple mixing based on capture time, an Initiator will normally use this knowledge to inform digital stitching techniques to align the multiple streams before running noise cancellation algorithms, but that is outside the Bluetooth specification.

Whilst the most common use of Presentation Delay for capture in unicast audio sources is the case of one microphone in each earbud or hearing aid, it is equally valid for single devices, which have multiple microphones. That includes stereo microphones and stereo headsets with a microphone in each can. In these devices an implementation could encode the two microphone signals into a single codec frame using channel allocation for multiplexing (see Section 3.11 below), or transmit them separately using Presentation Delay to ensure that the capture is aligned. In both cases, a value for Presentation Delay needs to be set to cover the audio processing and encoding time.

Where microphones are spaced further apart, the received data will not reflect the difference in audio path between their positions. The Initiator may need to adjust the relative timing of the incoming streams it receives if it wants to restore that information. As the Initiator schedules all of the incoming audio data, there is no specified reference point for these streams, as it is assumed that the implementation is aware of when each audio stream arrives.

It is important to understand that Presentation Delay is only applied at the Acceptor, regardless of whether it is acting as an Audio Sink or an Audio Source. In many cases an Acceptor will be acting as both. Typically, the values of Presentation Delay will be different for each

direction to cope with the fact that encoding audio data takes longer than decoding it. We will look in more detail at how Presentation Delay is used to influence latency and robustness in Chapter 4.

3.13 Remote controls (Commanders)

Bluetooth has always been a good candidate for remote control devices, but these are almost all simple remote controls, which are essentially a Bluetooth replacement for traditional infrared remote controllers. Hearing aids and earbuds make remote control an attractive option, as these devices are so small that they don't have room for many buttons, and even if they do, manipulating a button that you can't see (because it's on the side of your head or behind your ear) is a poor user experience. As most earbuds are used with phones, laptops or PCs, and the application generating the audio stream generally contains the controls that you use to pause it, answer a call or change volume, that's not a major issue. However, hearing aid users also need to control the volume of their hearing aids when they're just being used to amplify ambient sound.

To make it easier than fumbling for a button on the hearing aid, the industry has developed simple, keyfob-like remote controls that allow a user to easily adjust the volume or change the audio processing algorithms to suit their environments (these are known as preset settings). Even where a hearing aid contains Bluetooth technology, most of the time a user won't have an active Bluetooth link enabled, and getting your phone out of a pocket or bag, followed by finding the appropriate app is an inconvenient way to adjust volume. If you are troubled by a loud noise, it's quicker and easier to take your hearing aids out, which is not a good user experience.

The new broadcast capabilities of Bluetooth LE Audio will result in far more public infrastructure, where a user will need to navigate through multiple different broadcasts and decide which to receive. Not only is that difficult to do on a hearing aid or earbud, it involves relatively power-hungry scanning. To address these issues, the concept of a separate device was developed, which could provide volume control, discover and display available Broadcast Sources, discover keys for encrypted broadcasts, and allow hearing aid users to change their presets. It's also possible to use it to answer calls or control a media player. These devices take the Commander role, which is defined in CAP, but can also use the Broadcast Assistant role from BAP for discovering broadcasts, as well as being Content Control Clients. Most top level profiles introduce further names for additional Roles that these devices can assume. For the rest of the document, I'll use the Commander terminology, unless it's for the specific subset of a Broadcast Assistant.

The Commander is an important new addition to the Bluetooth LE Audio topology. The role can be implemented on a phone, either as a stand-alone application, or part of a telephony or audio streaming app. It can also be implemented in any device which has a Bluetooth connection to an Acceptor or a Coordinated Sets of Acceptors. That means you can

Section 3.13 - Remote controls (Commanders)

implement it in dedicated remote controls, smart watches, wrist bands and even battery cases for hearing aids and earbuds. Commanders can have a display to show textual information about broadcasts, to help you select them, or just volume buttons. Commanders work on a first-come, first-served basis, so you can have multiple Commanders, allowing you to use whichever comes to hand first when you want to change volume or mute your hearing aids. Because all of the functions are explicitly specified, it also means that multiple devices can implement them interoperably. In Chapter 12, we'll look at some of the ways in which they are likely to change the way in which we will use Bluetooth audio devices in the future.

-oOo-

Having covered these new concepts, we can now dive into the specifications to see how they work and how they enable new use cases for Bluetooth LE Audio.

Chapter 4. Isochronous Streams

If you've worked with Bluetooth® applications in the past, you've probably concentrated on the profiles and barely looked at the Core specification. That's possible because the Bluetooth Classic Audio profiles use well-defined transport configurations tied in with Core settings, so that there is not much need to understand what's happening underneath the profile or its associated protocol. With Bluetooth LE Audio profiles, that changes, as you have more potential to affect how the Core works than with any of the Bluetooth Classic Audio profiles.

In order to try and make the most flexible system possible, which would cope not only with today's audio requirements, but also the ones we haven't even thought about yet, the specifications had to allow a much greater degree of flexibility. To achieve that, a fundamental, architectural decision was made to split the audio plane and the control plane. What that means is that new isochronous physical channels have been defined, to carry the audio streams. These sit alongside, but are separate to the existing ACL links of Bluetooth LE. The isochronous physical channels in the Core let you build up a number of isochronous audio streams, which are capable of transporting all types of audio, from very low speech quality, up to incredibly high music quality. With one exception, which we'll see later on, Isochronous Streams contain no control information – they are purely for carrying audio. The accompanying ACL channels are used to set up the Isochronous Streams, turn them on, turn them off, add volume and media control, along with all of the other features that we need, using the standard GATT²⁶ procedures that are part of Bluetooth LE.

In order to provide the flexibility that is needed for different latencies, different audio quality and different levels of robustness, developers need to be able to control the way these Isochronous Streams are configured. That's done quite high up in the profile stack of the Generic Audio Framework. It means that when you start working on Bluetooth LE Audio applications, even if you're just using the top level profiles, you still need to know a fair amount about how the underlying Isochronous Streams work. That's different from what you would have experienced in most previous Bluetooth applications. To help understand the overall architecture of Bluetooth LE Audio, we need to look at how those Isochronous Streams were developed, what they do, and how to use them.

4.1 Bluetooth LE Audio topologies

Up until now the Bluetooth specification has largely been concerned with peer-to-peer connections: a Central²⁷ device makes a connection to a Peripheral device, and they exchange

²⁶ The Generic Attribute Profile is part of the Core and defines the procedures which are used with the Attribute Protocol in Bluetooth LE.

²⁷ From December 2020, the Bluetooth SIG, like many other standards organisations, implemented a policy of replacing words which are considered to have negative connotations. That means that the specifications no longer use the traditional engineering terminology of Master and Slave when

data. It's a very constrained topology. Various companies have developed proprietary extensions to add flexibility, as we've seen with True Wireless Stereo earbuds, but Isochronous Streams were developed to cope with a much wider range of topologies than even these proprietary solutions could provide. As well as connecting a mobile phone to a pair of headphones or a single speaker, Bluetooth LE Audio needed the ability to send separate left and right signals to a left earbud and a right earbud. It also needed to be able to send the same information to more than one set of earbuds and scale up the number of devices and streams. These are all covered in the current Bluetooth LE Audio specifications. They also provide the foundation for further audio applications, with work already underway on spatial audio, surround sound, and voice recognition capabilities.

There are two types of Isochronous Stream – unicast and broadcast. Unicast connections, known as Connected Isochronous Streams (CIS), are the closest to existing Bluetooth audio use cases. “Connected” in this sense means that they are transferring audio data between two devices, with an acknowledgement scheme between the two devices to provide flow control. Connected Isochronous Streams have an ACL control channel that is up and running throughout the lifetime of the CIS which is carrying the audio data.

A very similar structure of Broadcast Isochronous Streams (BIS) is used for broadcast, but there's a major difference. With broadcast, a device that transmits the Isochronous Streams has no knowledge of how many devices may be out there receiving the audio. There's no connection between devices and no need for an ACL link. At its simplest, broadcast is purely promiscuous. However, it is possible to add control links to broadcast. At the Core level, there is a clear distinction between Connected and Broadcast Isochronous Streams, which is based on whether data is acknowledged or not. However, many applications will switch between unicast and broadcast to fulfil different use cases, without the user being aware of what is happening. But for the time being, we'll concentrate on the basics.

Broadcast allows multiple users to hear the same thing, in the same way as FM radio or broadcast TV. The requirements for Bluetooth LE Audio broadcast capability were initially driven by the hearing aid application of telecoils, where multiple people wearing hearing aids in a public venue can listen to the same signal. Telecoils can provide good audio quality, but are mostly used for speech. With the higher quality possible with Bluetooth LE Audio, along with a significantly lower installation cost, the industry envisaged a much wider range of applications and usage, including the opportunity to manage audio services in commercial and hospitality venues. Traditional telecoil locations like conference centres, theatres and places of worship; public information, such as flight announcements, train departures and bus times would become accessible to everyone with a headset and earbud, not just people wearing

describing communications, but have replaced them with Central and Peripheral. A full list of these changes can be found at www.bluetooth.com/languagemapping/Appropriate-Language-Mapping-Table.

Section 4.2 - Isochronous Streams and Roles

hearing aids. Broadcast is also applicable for more personal applications, where a group of people can listen to the same TV, or share music from their mobile phones. That last example shows how Bluetooth LE Audio applications can invisibly switch the underlying protocols back and forth. If you are streaming music from your phone to your earbuds, it's probably using a Connected Isochronous Stream. When your friends come along and you ask them "Do you want to listen to this too?", your music sharing application will switch your phone from a private, unicast connection to an encrypted broadcast connection, so that you can all hear the same music, whether that's on earbuds, hearing aids, headphones or speakers. At the application layer, listening to the music and sharing it with any number of other people should be seamless – the users don't need to know about broadcast or unicast. But there will be a lot going on underneath during that transition.

The building blocks for these different use cases are essentially the same. There are some differences between the Connected Isochronous Streams that are used for unicast use cases and Broadcast Isochronous Streams, which are used for the broadcast use cases, but the underlying principles are very similar. We're now ready to see how they're all put together, which means diving into the Core.

4.2 Isochronous Streams and Roles

The Isochronous Streams feature in the Bluetooth® Core 5.2 release is a fundamentally new concept within Bluetooth LE. If you're familiar with Hands-Free profile or A2DP, you'll know that they have quite a constrained topology. HFP has a bidirectional one-to-one link, typically between a phone and a headset or Hands-Free device. It has two roles: an Audio Gateway, and a Hands-Free device. A2DP is an even simpler unicast link, specifying a Source device that generates audio and a Sink device, which can be your headphone, speakers, amplifier or a recording device, which receives that audio.

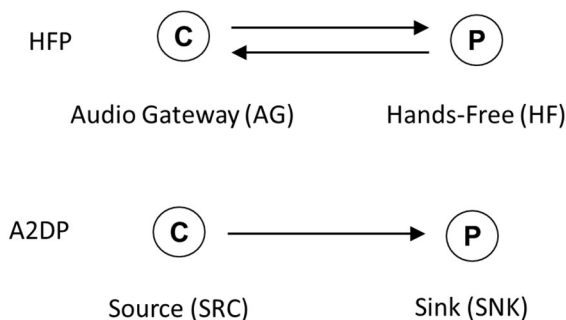


Figure 4.1 Bluetooth Classic Audio topologies

Bluetooth LE Audio is built on the fundamental asymmetry that exists within the Bluetooth LE specification, where one device - the Central device, is responsible for setting up and controlling the Isochronous Streams. The Central can connect to a number of Peripheral devices, using Isochronous Streams to send and receive audio data. The asymmetry means

that the Peripheral devices can be much lower power. For CISes, Peripherals now have a say in how the Isochronous Streams are configured, which will affect the audio quality, latency and their battery life, giving them more control over the audio streams than with Bluetooth Classic Audio profiles. For BISEs, the Central makes all of the decisions about what it transmits, with the Peripheral left to decide which Broadcast Isochronous Streams it wants to receive. Both act independently of each other, although it is possible to let them communicate to enable more complex applications.

Repeating what we said in the terminology overview in Section 3.3, as we move up the stack of the Bluetooth LE Audio specifications, we'll come across a number of different names for the roles which devices perform. In the Core, they are defined as Central and Peripheral devices. In the BAPS set of specifications they're called Clients and Servers, and in CAP they become Initiators and Acceptors. The Initiator role always exists in a Central device which is responsible for scheduling the Isochronous Streams. Acceptors are the devices that participate in these streams. There is always one Initiator, but there can be multiple Acceptors.

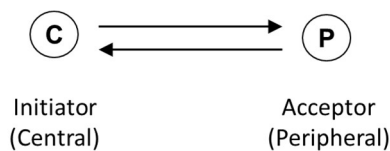


Figure 4.2 Bluetooth LE Audio roles

When we move up into the top level profiles, there's an avalanche of new role names, with Senders, Broadcasters and Receivers. I'm going to ignore all of those and use the Initiator and Acceptor names for most of time (even though they're technically roles), because I think they best explain the way that Bluetooth LE Audio Streams work. When there's no Audio Stream involved, which is the case with the Control specifications, I'll drop back to using Client and Server.

One thing I'd like to point out at this stage is that other than for broadcast, either device can act as an audio source, generating audio data, or as an audio sink, receiving that data. Both Initiators and Acceptors can be sources and sinks at the same time and they can each contain multiple Bluetooth LE Audio sinks and sources. The concept of who generates the audio and who receives and renders it is orthogonal to the concept of the Initiator and the Acceptor. The important thing to remember is that the Initiator is the device that is responsible for working out the timing of every transmission of audio data that is sent between itself and the Peripherals. That task is called scheduling. The Acceptor is the device that accepts those streams. That concept applies both in unicast and broadcast. A unicast Acceptor can also generate audio data, such as capturing your voice from your headset's microphone, but the Initiator is responsible for telling it when it needs to send that data back. That same principle applies regardless of whether the CIS is used as a unidirectional or a bidirectional stream.

Section 4.2 - Isochronous Streams and Roles

As the Initiator role is far more complex than the Acceptor role, Initiators are normally devices like phones, TVs and tablets which have bigger batteries and more resources. The scheduling has to take account of other demands on the Initiator's radio, which may include other Bluetooth connections, and often Wi-Fi as well. That's a complex operation which is handled by the chip designers. But, as we'll see later on, the Bluetooth LE Audio profiles give applications a fair amount of scope to influence that scheduling, which is why developers need to have a clear idea about how Isochronous Streams work.

For unicast Bluetooth LE Audio, we have a lot of flexibility in terms of topologies, which are depicted in Figure 4.3. We can replicate the same topologies that we had in HFP or A2DP, where we have a single device - typically your phone, and a single Peripheral device, such as your headset, with an audio link between them. Moving up from that, Bluetooth technology can now support an Initiator that talks to two or more Acceptors. The main application for that is to allow your phone to talk individually to a pair of left and right earbuds or hearing aids. They no longer need to be from the same manufacturer, as Bluetooth LE Audio is an interoperable standard.

We can extend this by adding additional unicast streams to support more than one pair of earbuds, or to connect multiple speakers to support surround sound systems, with the example of Figure 4.3 showing the addition of a central woofer unit.

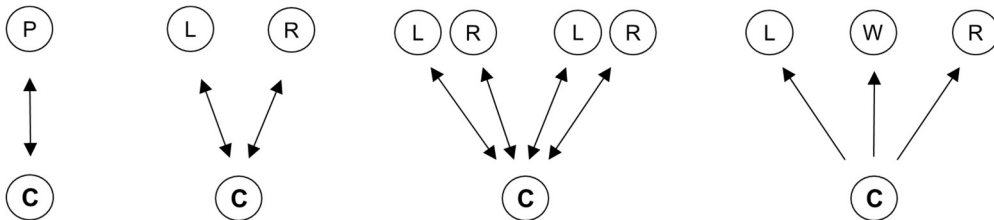


Figure 4.3 Unicast audio topologies

In theory, the specification can support up to 31 separate unicast Isochronous Streams, which could connect 31 different devices. That's not actually realistic for audio, as we start to run out of bandwidth after three or four streams. The reason for the limit of 31 streams in the Core specification is that the Isochronous Streams feature was designed to support many different time critical applications – not just audio. Some of those need far less bandwidth than audio does. When we look at the LC3 codec, we'll discover some of the trade-offs we have to make between latency, audio quality and robustness, which place a limit on the number of audio streams we can actually support.

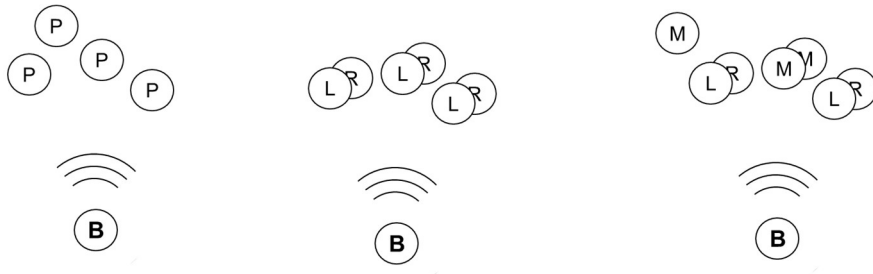


Figure 4.4 Broadcast audio topologies

That airtime limitation on the number of streams that unicast can support is one of the reasons to use broadcast. As Figure 4.4 shows, a single Broadcast Source can talk to multiple Acceptors, which will often be configured as pairs of devices, receiving either a single mono channel or separate left and right audio channels. Depending on the desired audio quality (which typically determines the sampling rate and hence the airtime usage and maximum number of streams), a Broadcast Source may be able to offer greater functionality, such as transmitting simultaneous audio streams in different languages. These are the trade-offs that need to be understood when designing a Bluetooth LE Audio application, which we'll cover in more detail in the following chapters.

4.3 Connected Isochronous Streams

To explain the Core Isochronous features, we'll start with unicast and Connected Isochronous Streams, which are known as CISes. Their structure is quite complex, but it's built on some very simple principles. I'll start by describing how Connected Isochronous Streams were designed, to explain the component parts and how they work, then look at how Broadcast is different. That gives you the foundations to move up into the Generic Audio Framework, where we put the Isochronous Streams to work.

4.3.1 The CIS structure and timings

When you design for digital audio, you generally have the constraint that you're going to be sampling the incoming audio at a standard, consistent rate. Once the incoming audio is sampled and encoded, it's sent down to the Bluetooth transmitter to send to the receiving device. The system is repetitive, the audio data is time bounded, and transmissions have a constant interval between them which is called the Isochronous Interval (ISO_Interval). The start of each Isochronous Interval in a CIS is called its Anchor Point. As Figure 4.5 shows, the transmission starts off with an Initiator sending a packet containing audio data (D) to an Acceptor. When the Acceptor receives it, it sends back an acknowledgement, and that process is repeated on a regular basis. The third data packet in Figure 4.5 has no acknowledgement, so the Initiator would presume it had not been received.

Section 4.3 - Connected Isochronous Streams

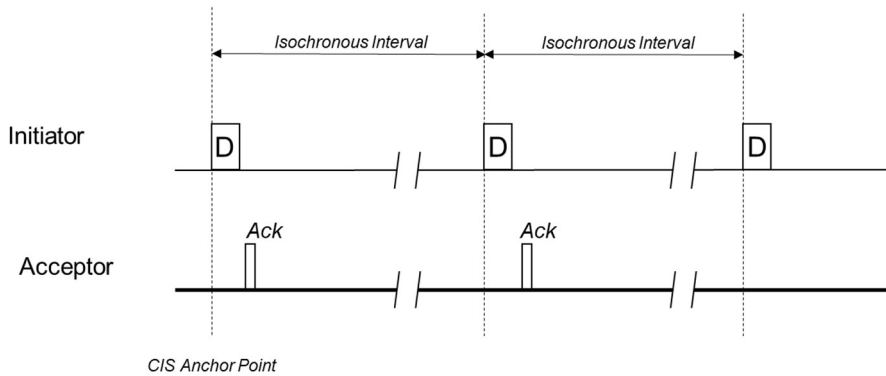


Figure 4.5 Simplified unidirectional audio transfer

Most modern codecs are optimised to run at a frame rate of 10 milliseconds, i.e., they sample 10 milliseconds of audio at a time, which provides a good compromise between audio quality and latency. That's the preferred setting for LC3, which is the mandatory codec for Bluetooth LE Audio. Unless I specify otherwise, we'll be assuming a 10 millisecond sampling interval is used throughout this book, so the Isochronous Intervals will always be 10ms, or multiples of 10ms.

4.3.1.1 Isochronous Payloads

The structure of the data in the PDU of the air interface packet (shown as “D” in Figure 4.5), which is sent between devices is very simple. It is illustrated in Figure 4.6.

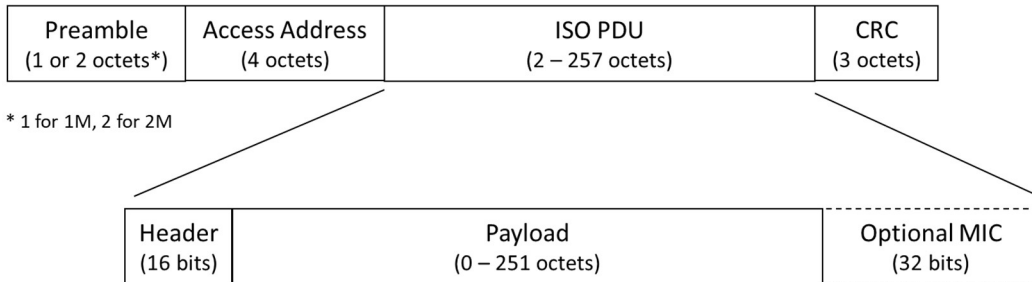


Figure 4.6 Bluetooth® LE Link Layer packet format

The encoded ISO PDU is preceded by a preamble and Access Address, and followed by a CRC. These add 8 or 12 octets when transmitting over an LE 1M PHY²⁸, (depending on

²⁸ PHY refers to the Physical layer, and specifically the choice of symbol rate, which corresponds to the number of bits which can be transmitted each second. Currently most Bluetooth LE products use a symbol rate of 1 million symbols per second. In contrast, most Bluetooth LE Audio products will use the enhanced symbol rate of 2 million symbols per second, as that allows higher quality codec parameters to be used. The trade-off is a slight reduction in range.

whether there is a Message Integrity Check (MIC)²⁹ included with the ISO PDU payload), and 9 or 13 octets when using a LE 2M PHY. If a MIC is included, it reduces the maximum Payload size. The CRC is calculated across entire ISO PDU (i.e. the Header, Payload and MIC).

For a CIS, the ISO PDU is called the CIS PDU, and its structure is shown in Figure 4.7.

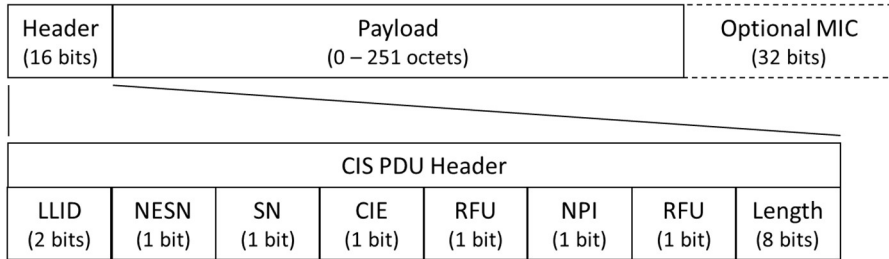


Figure 4.7 ISO PDU format and header for a CIS

The ISO PDU has a header, followed by a payload of up to 251 octets. If the audio needs to be encrypted, there’s an optional MIC at the end of that packet. In the CIS PDU header, there are five control elements, as well as 8 bits defining the length of the packet:

- the LLID (Link Layer ID), which indicates whether it is framed or unframed
- NESN and SN, the Next Expected Sequence Number and Sequence Number, which are used for acknowledgments and flow control at the Link Layer level,
- CIE, the Close Isochronous Event bit, and
- NPI. The Null Payload Indicator, which indicates the payload is a null PDU, identifying that there is no data to send.

4.3.1.2 Subevents and retransmissions

We’ll cover the control bits in the ISO PDU header as they become relevant to the explanation of how CISes work. Before that, we need to look at the structure of a Connected Isochronous Stream. We’ve already talked about the Isochronous Interval, which is the time between successive Anchor Points of a CIS. The Anchor Point is the point where the first packet of a CIS is transmitted by the Initiator and the start of each successive Isochronous Interval. Within a CIS, we can retransmit the CIS PDU multiple times if required, as the CIS structure supports multiple Subevents. Each Subevent starts with the transmission from the Initiator, followed after an interframe spacing (T_IFS) of 150µs³⁰ by the response from the Acceptor,

²⁹ Message Integrity Check. A value calculated from the payload contents to detect if it has been corrupted.

³⁰ The recent Bluetooth® Core 6.0 release allows the T_IFS to be reduced to values as low as 50µs.

Section 4.3 - Connected Isochronous Streams

and ending at the final point of the expected response from an Acceptor. All of the Subevents within a CIS form a CIS event, which starts at the Anchor Point of the CIS and finishes at the reception of the last transmitted bit received from the Acceptor. That's a lot easier to understand in a diagram, which is what is shown in Figure 4.8

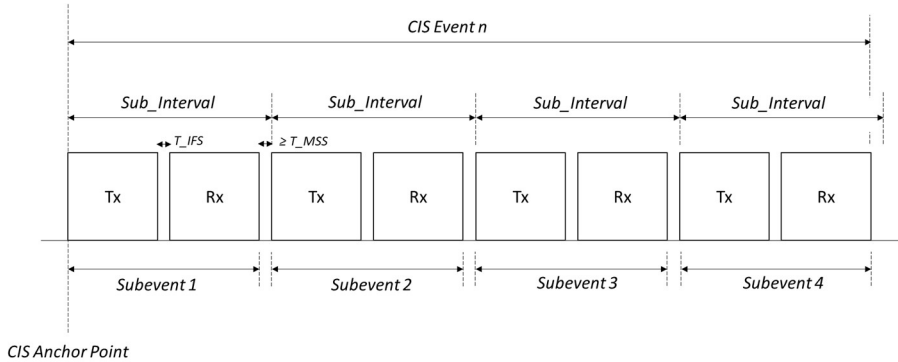


Figure 4.8 Events, Subevents and Sub_Intervals

The time between successive Subevents is defined as the Sub_Interval spacing, which is the maximum duration of the Subevent for that CIS, plus the Minimum Subevent Space (T_{MSS}), which is defined as being at least $150\mu s$. The Sub-Interval spacing is determined when the CIS is configured and does not change for the lifetime of the CIS.

4.3.1.3 Frequency hopping

An important reason for defining Subevents is that Bluetooth LE Audio changes the frequency of the transmission channel on every Subevent, as illustrated in Figure 4.9, to protect against interference.

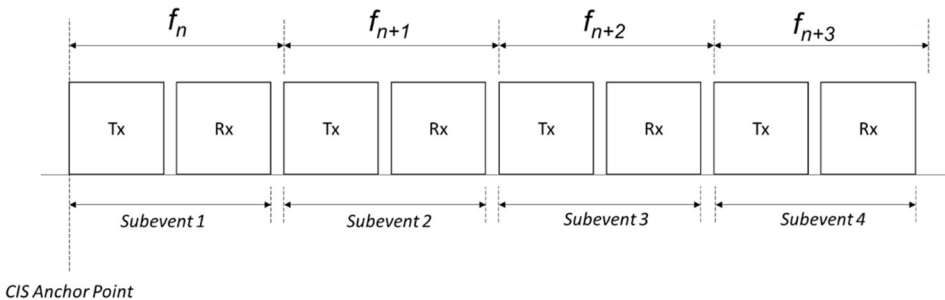


Figure 4.9 Frequency hopping per Subevent

The Bluetooth® Core 5.2 specification introduced a new channel selection algorithm, which is more efficient than the one originally defined in Bluetooth® Core 4.0. This is applied to each Subevent. If a Subevent is not transmitted, then the channel hopping scheme assumes that it has been and moves to the next frequency channel for the following Subevent.

4.3.1.4 Closing a Subevent

When we looked at the isochronous PDU header, we saw that there's a Close Isochronous Event bit – the CIE. That's used by the Initiator to signify that it has received an acknowledgment from an Acceptor confirming that its packet was successfully received by the Acceptor, so that it will stop further retransmissions of that particular PDU. In Figure 4.10, the Initiator has sent out its first PDU and had an acknowledgment back, so it then sends a packet with the CIE bit set to 1, to tell the Acceptor that there will be no further transmissions, as it has successfully received the response and is closing the event. It would not normally bother to include the audio data in that transmission, so it would also set the Null Packet Indicator bit, allowing it to shorten the transmitted packet. (Figure 4.10 shows the duration of the original CIS Event for comparison.)

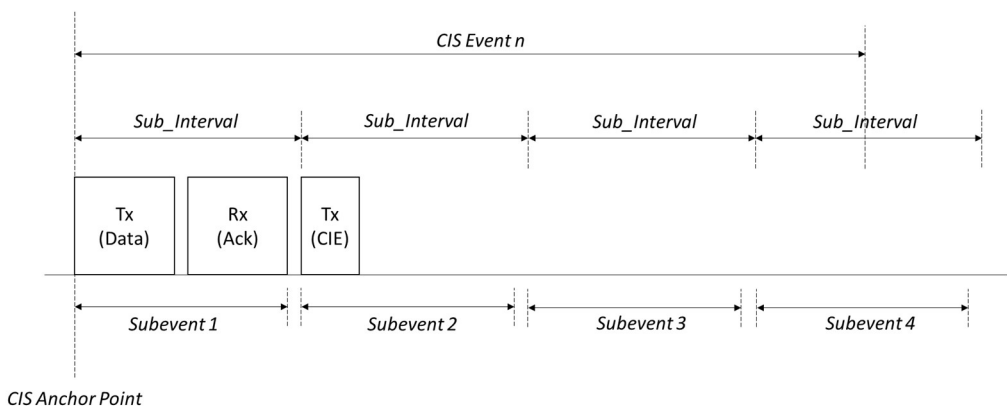


Figure 4.10 Closing a CIS event early with the CIE bit

This allows the Acceptor to go to sleep until the next Isochronous Interval. The Initiator can use the time to do other things. As many Initiators will also be interacting with other Bluetooth devices, and possibly sharing their radio and antenna with Wi-Fi, that extra time can be useful. For the Acceptor, turning its receiver off until it's ready to do something can bring a significant power saving.

If the Acceptor does not receive the header with the CIE bit, it may continue to listen for data in each scheduled Subevent, but will not receive any packets from the Initiator to respond to.

4.3.2 Controlling audio quality and robustness

Having covered the basic timing of transmissions in a CIS, we can now look at the parameters which are used to control the quality of the audio and the robustness of the link. For many audio applications, latency is important. For some applications, such as when you are listening to a live stream, it is important to minimise the latency, particularly if you can hear the ambient sounds as well. On the other hand, if you're streaming music through your phone and can't hear or see the source, latency doesn't matter that much. Other applications, such as gaming, have different priorities, and if you're listening to audio while watching a film, lip-synch becomes important.

Section 4.3 - Connected Isochronous Streams

The Basic Audio Profile (BAP) can set many of the parameters which affect audio quality and latency, by using the `LE_Set_CIG_Parameters` HCI command. We'll look at how it makes those choices in Chapter 7, but for now, we need to understand how the Isochronous Channel structure can be configured. The key items that an application can request in order to influence the latency and robustness are:

- The Maximum Transport Latency, which sets the maximum time that an Initiator can spend transmitting the SDUs that a controller receives for a particular CIS.
- The Maximum SDU size³¹ for both directions of the CIS
- The SDU interval for both directions

The Maximum Transport Latency affects the overall end-to-end latency, although it is only one element of it. Whilst many applications will want to minimise latency, in the real world of wireless you also need to address the inherent fragility of a wireless link where packets can be lost. To ensure sufficient robustness, (which translates into rendered voice and music streams without drop-outs, clicks and silence), we need to use a variety of techniques to help ensure that audio data gets through in an acceptable timeframe. These may result in an increase in latency.

4.3.2.1 Flush Timeout and Number of Subevents

The three parameters listed above are inputs to the Controller. The Controller takes them and uses them to calculate three parameters that affect the robustness of the Bluetooth LE Audio link for that CIS, which are:

- NSE - the Number of Subevents. This specifies the number of Subevents which will be scheduled in each Isochronous Interval. They are used for the initial transmission of a CIS PDU and its subsequent retransmissions. They may not all be used, but it is a fixed number that are scheduled.
- FT – the Flush Timeout. The Flush Timeout defines how many consecutive Isochronous Intervals can be used to transmit a PDU before it is discarded. The point at which it is no longer transmitted is called the Flush Point.
- BN – The Burst Number, which is the number of PDUs supplied for transmission in each CIS event.

These can be quite difficult to grasp, so it's useful to look at some simple examples.

The Number of Subevents (NSE) is the most straightforward of the three. It is simply the number of opportunities to transmit an Isochronous PDU which are available within each Isochronous Interval. In the simplest example, where only one PDU is supplied for

³¹ An SDU can contain more than one codec packet.

transmission in each Isochronous Interval, the PDU will be transmitted in the first Subevent, and can then be retransmitted a maximum of (NSE-1) times in subsequent Subevents in the same Isochronous Interval. If it is a unidirectional CIS, once the PDU's transmission is acknowledged by the Acceptor, the Controller can set the Close Isochronous Event (CIE) bit in the header of its next scheduled transmission of that PDU (which can be a CIS Null PDU with no data), and any remaining Subevents in that CIS Event become free airtime for other radio applications.

That is the case where Flush Timeout is 1, as the Flush Point then coincides with the end of the CIS Event for the Isochronous Interval. This simple case is illustrated in Figure 4.11. For the sake of clarity, the following examples only involve one Acceptor.

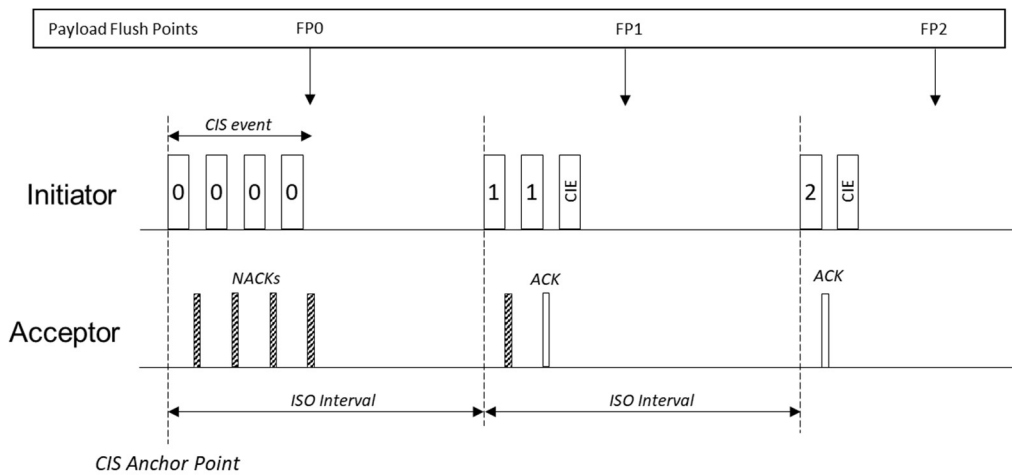


Figure 4.11 A unidirectional CIS with NSE = 4 and FT = 1

In this example, NSE is set to 4, so there are four opportunities for each packet to be transmitted. In the first CIS Event, none of the four attempts are successful, so packet P0 is flushed. The Acceptor will need to try to reconstruct it using some form of Packet Loss Correction.

The second packet (P1) succeeds after the second attempt, after which the Initiator closes the Event. The third packet (P2) succeeds first time.

If the Flush Timeout is increased, then the transmission of a packet can continue over more Isochronous Intervals. Figure 4.12 illustrates an example where NSE remains at 4, but the Flush Timeout is increased to 3.

Section 4.3 - Connected Isochronous Streams

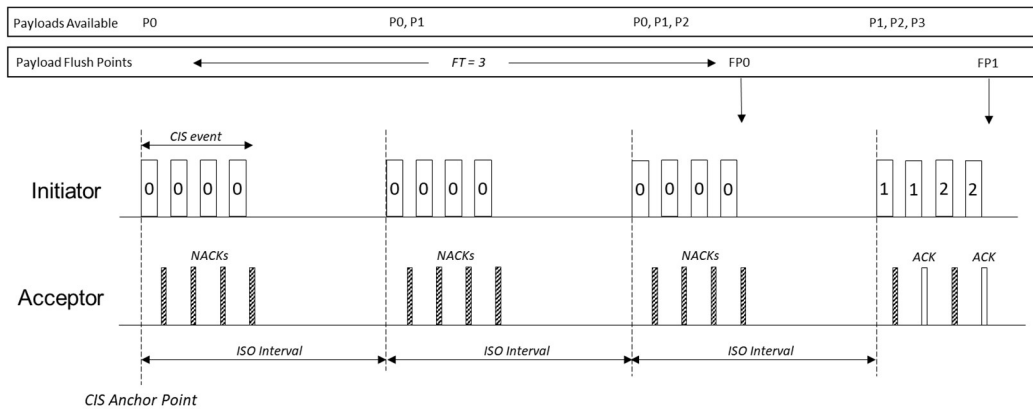


Figure 4.12 An example of $FT = 3$ and $NSE = 4$

This illustrates a problem. If you are only using the two parameters – NSE and FT, it allows a packet to dominate the transmission slots until it reaches its flush point. In Figure 4.12, the first payload, P0, which is having problems getting through, occupies all of the Subevents in the first three Isochronous Intervals, leaving P1 and P2 waiting until after the Flush Point FP0. Although this situation should not be common, it can leave subsequent payloads more exposed until enough of them get through to bring the system back into equilibrium.

4.3.2.2 Burst Number

The way to address this problem is to allow more than one payload to be transmitted in a single Isochronous Interval, so that Subevents in each Isochronous Interval can be shared between more than one PDU and not be used exclusively by one of them. This is made possible by taking advantage of Burst Number (BN), which is the number of payloads supplied for transmission in a CIS event. In the examples above, only one payload has been delivered to the controller in a CIS Event, which is the situation where the cadence of SDU and PDU generation is the same as the Isochronous Interval – meaning that one encoded 10ms audio frame becomes available for each 10ms Isochronous Interval. If we want to make use of BN and we're continuing to sample the incoming Audio Channel every 10ms, we need to increase the Isochronous Interval to a multiple of that, so that we have more payloads available in each Isochronous Interval. It means that by addressing one problem, we are potentially creating another, which is increasing latency.

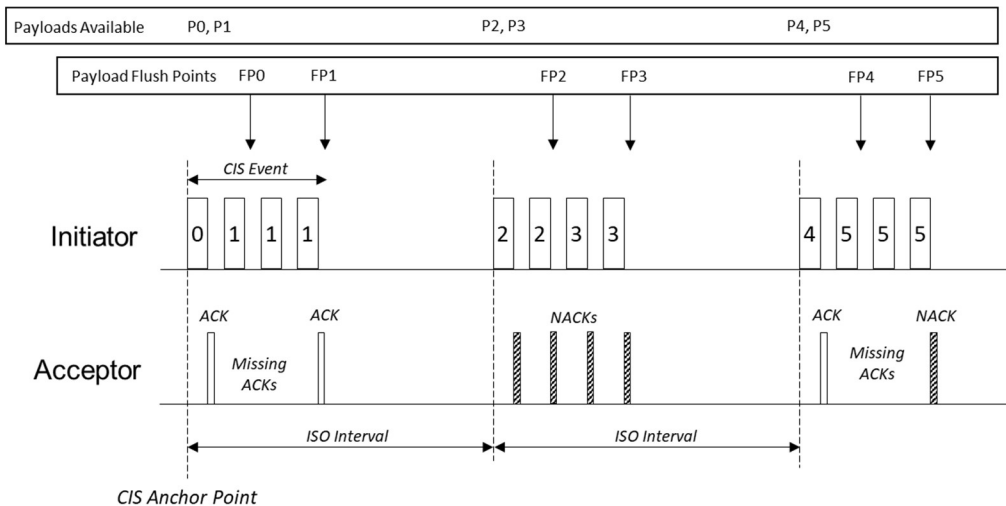


Figure 4.13 The effect of Burst Number = 2, with NSE = 4 and FT = 1

In Figure 4.13, the Isochronous Interval has been doubled, allowing two payloads to be supplied in each interval. The combination of the 10ms codec frame and 20ms Isochronous Interval means the Initiator has two payloads available to transmit within each Isochronous Interval. A consequence of this is that there are now two flush points in each Isochronous Interval. In our simple example, each Flush Point occurs after two Subevents. For more complex combinations of FT and NSE parameters, the location of the Flush Points can be calculated from the equations in the Core [Vol 6, Part B, 4.5.13.5].

Returning to Figure 4.13, we see the Initiator sending packet P0 in the first Subevent, which is acknowledged, so the Controller immediately moves on to transmit packet P1, transmitting it three times before it is acknowledged.

In the second Isochronous Interval, packet P2 is transmitted, but the Acceptor sends NACKs to indicate errors in the received packet. As the Flush Timeout is set to 1 and BN=2, the Flush Point for packet P2 is in the same Isochronous Interval, coming after two further transmission attempts, neither of which have been successfully received by the Acceptor.

At that point, the Initiator starts transmitting P3, although once again, in this example, the Acceptor responds with NACKs to indicate a problem with the packets it received. After two attempts, P3 is flushed by the Initiator.

In the final Isochronous Interval of Figure 4.13, P4 is successfully transmitted, leaving three opportunities for P5, all of which are unsuccessful. In each case, the Initiator will attempt to transmit the PDUs in every available Subevent before that PDU's Flush Point. After each acknowledged transmission it will move on to the next available packet or, if there are no further packets available, close the Isochronous Event.

Section 4.3 - Connected Isochronous Streams

In this example, P2, P3 and P5 would be discarded. In real life, we'd expect a much better rate of acknowledgement – this is just an example to illustrate the principle.

As a final example, Figure 4.14, shows the effect of increasing the Flush Timeout to 2, with a Burst Number of 2, which gives the opportunity to transmit in two consecutive Isochronous Intervals. In this example, no packets are flushed.

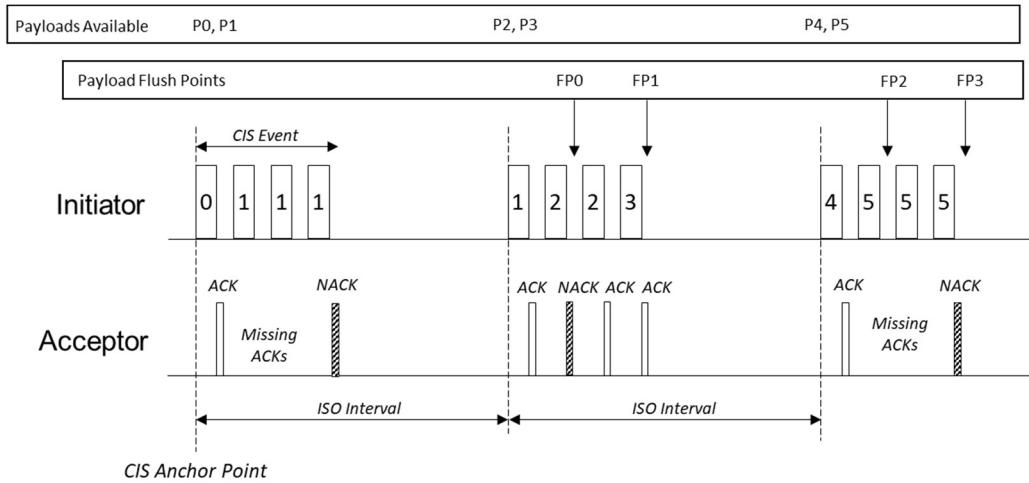


Figure 4.14 Example of NSE=4, BN=2 and FT=2

Using Flush Timeout and Burst Number can be very useful to provide more retransmission opportunities, spanning multiple Isochronous Intervals. They are particularly useful if you're in a noisy environment. However, they have an effect on latency. Every increment of Flush Timeout increases latency as the retransmissions are spread across more Isochronous Intervals, whilst Burst Number increases the duration of the Isochronous Intervals. Note that Burst Number is only applicable to multiple payloads arriving in an Isochronous Interval. It can't be used to solve the problem of a single payload hogging transmission opportunities which we saw in Figure 4.12, but that is the primary technique to deal with interference. So, it is always a trade-off.

Burst Number and Flush Timeout cannot be set directly by the Host. The Host and the applications running on it are limited to setting values for the Maximum Transport Latency, the Maximum SDU size and the SDU interval. BN and FT are then calculated in the Controller, which takes into account any other radio requirements in the chip.

The host can request a Retransmission Number (RTN) using the LE Set CIG Parameters HCI command, which is NSE - 1, but the Controller only treats it as a recommendation.

It is useful to have an understanding of the potential effects these parameters have on the Isochronous Channel structure. We can find that in Table 3.15 of TMAP, which provides examples of how a Controller might interpret the Host values for a CIS to suit different

operating conditions, such as prioritising airtime for coexistence or minimising latency. The exact allocation of parameters is always down to the algorithms in the scheduler within the Controller, which will be set by the chip supplier. This will take account of other resource requirements, such as other Bluetooth connections and concurrent Wi-Fi streams, before allocating values for the requested Isochronous Channels.

4.3.3 Framing

The other parameter in a CIS PDU header which needs to be understood is the pair of LLID (Link Layer ID) bits, which indicate whether the CIS is framed or unframed. Unframed refers to the case where a PDU consists of one or more complete codec frames. It is used where the Isochronous Interval is an integer multiple of the codec frame length. In contrast, framed is where you have a mismatch between the codec frame length and the Isochronous Interval, which results in a codec frame being segmented across multiple SDUs. This starts to get complex, but is important where an Initiator may need to support Bluetooth connections which have different timings. We'll revisit that when we look at ISOAL - the Isochronous Adaptation Layer, which has been designed to cope with this mismatch.

If the packet is a CIS Null PDU, which is indicated by the Null PDU Indicator (NPI) bit in the ISO PDU header, the LLID bits are treated as RFU bits and should be ignored.

4.3.4 Multiple CISes

Having covered all of the features of a single, unidirectional CIS, the next step is to add more of them. The most common application for this is when an Initiator is sending stereo audio streams separately to a left and a right earbud. In this case, the Initiator will set up separate Isochronous Streams with two different Acceptors. In Figure 4.15 we can see that it's a straightforward extension of what we've seen before - the Initiator transmits and receives an acknowledgment from the first Acceptor, then repeats this for the second Acceptor.

An important point to note is that although the left and right Audio Channels are sampled at the same time, the ISO PDUs are sent serially.

Section 4.3 - Connected Isochronous Streams

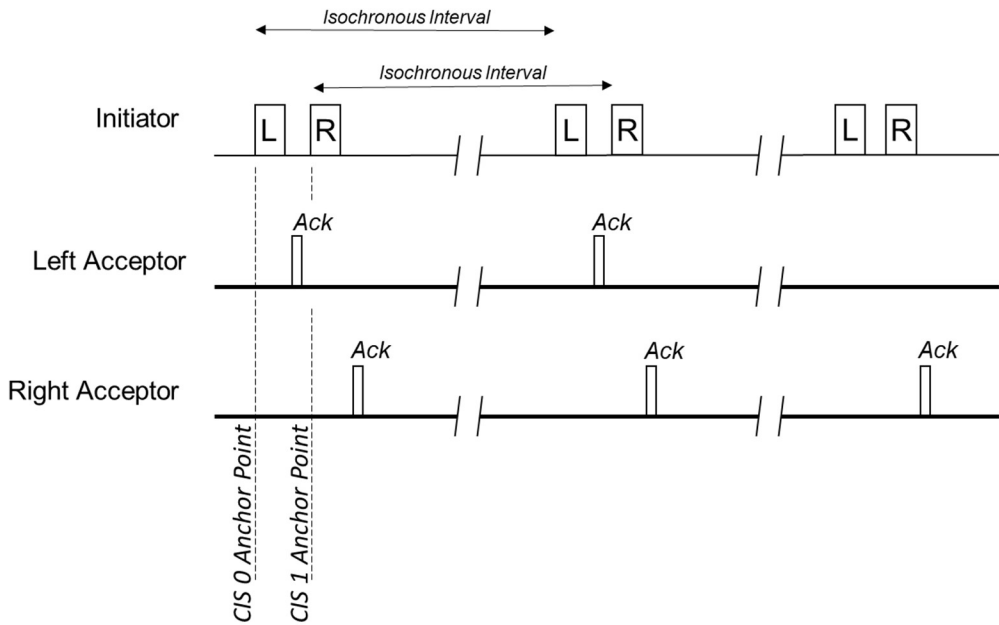


Figure 4.15 Timings for CISEs to two Acceptors

Note that where we have more than one CIS, they always have the same Isochronous Interval. Their Anchor Points are different, as data is sent serially, but for each CIS their Anchor Points are the same ISO Interval apart. Each Anchor Point represents the point of the first transmission from an Initiator to an Acceptor for that CIS.

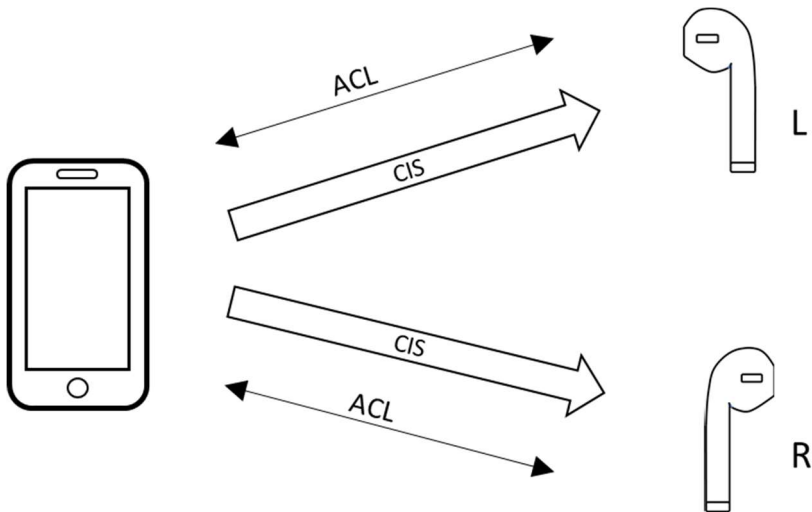


Figure 4.16 Multiple CISEs and associated ACL links

As Figure 4.16 shows, each CIS has an associated ACL link. The ACL link always needs to be present because it is used to set up the CIS and control it. If there are multiple CISEs between an Initiator and an Acceptor, they can share the same ACL. If the ACL is lost for any reason, all CISEs associated with that ACL are terminated. The application then needs to

decide what to do with any remaining CISes established with other Acceptors within that CIG³².

When an Initiator is scheduling two or more audio channels, there are two options for how they are transmitted, regardless of whether they are connected to one or multiple Acceptors. The obvious approach is to transmit them sequentially so that you send all of the data on CIS 0, and then all the data on CIS 1, continuing until the Initiator has worked through all of the packets for both CISes. This is illustrated in Figure 4.17, where we have an NSE of 3. It shows all of the Subevents within CIS 0 being transmitted before the Subevents for CIS 1.

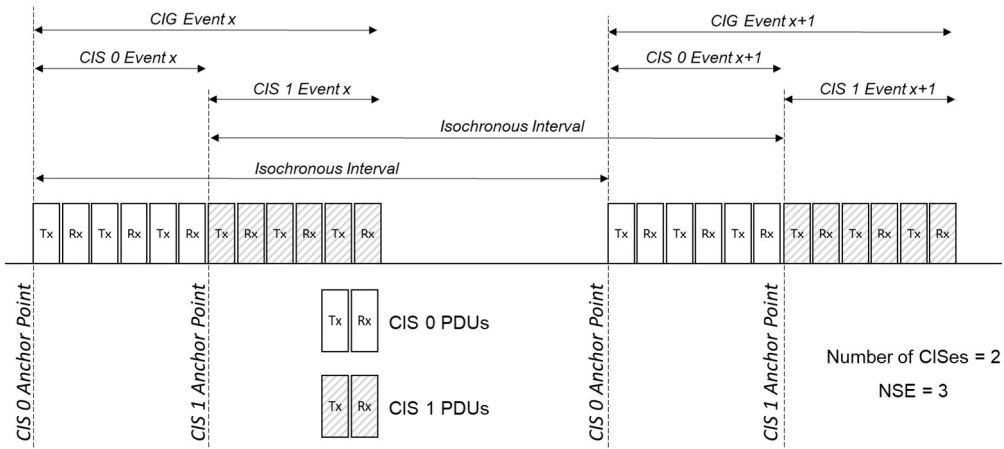


Figure 4.17 Sequential arrangement of two CISes

The disadvantage of this approach is that if you have a reliable connection, where the first transmission of the packet is acknowledged, you end up with gaps between each CIS. In devices like phones, which are often sharing the Bluetooth and Wi-Fi radios, that's wasted airtime that could be used for something else. To address this, the Controller can choose to interleave CISes as an alternative approach, as shown in Figure 4.18.

³² An example is where the link quality to left and right earbuds differ significantly, so that one of them is lost. The application may decide to continue to stream to the earbud with the good link, (possibly changing to a mono stream), whilst it attempts to recover the connection to the other earbud.

Section 4.3 - Connected Isochronous Streams

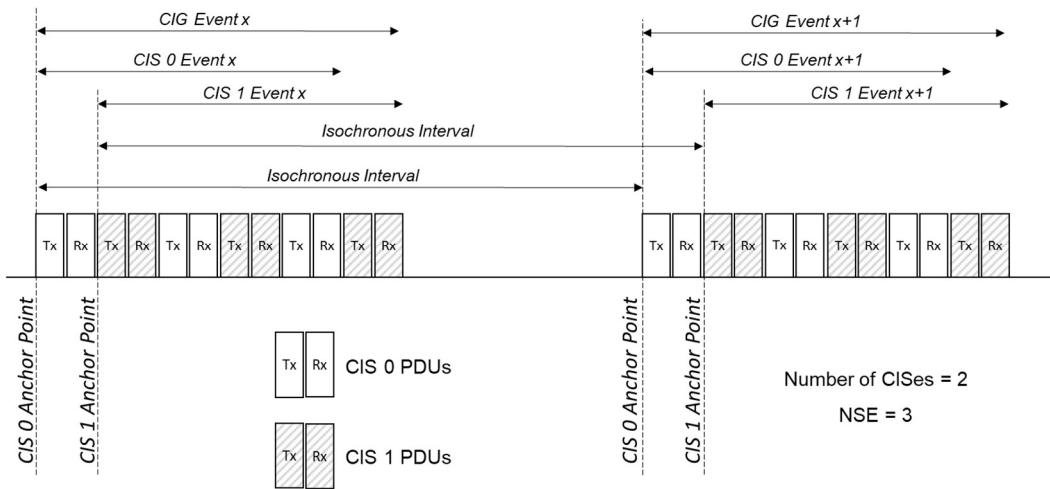


Figure 4.18 Interleaved arrangement of two CISes

In this case, each Subevent for CIS 0 is followed by a Subevent for CIS 1, and so on for the remaining CISes. The diagram repeats the example of Figure 4.17, having an NSE of 3, and shows CIS 0 transmitting and receiving its first Subevent, followed by CIS 1. If both Acceptors receive those first transmissions and acknowledge them, they can close their CIS Events. It results in a much greater gap after their transmissions, freeing up airtime which can be used for other purposes.

4.3.5 Bidirectional CISes

The multiple, unidirectional connected Isochronous Streams we've defined above replicate the use case of A2DP. For earbuds and many other audio applications, we want to get data back as well, requiring bidirectionality. One approach would be to set up a second CIS in the opposite direction, where we would use one CIS to transmit audio from phone to earbud and the other to transmit from earbud to the phone. However, that's inefficient. The Core specification provides an optimisation by adding return data into the Acceptor's acknowledgement packets. It's still a single CIS, but it is now being used for two separate Audio Streams.

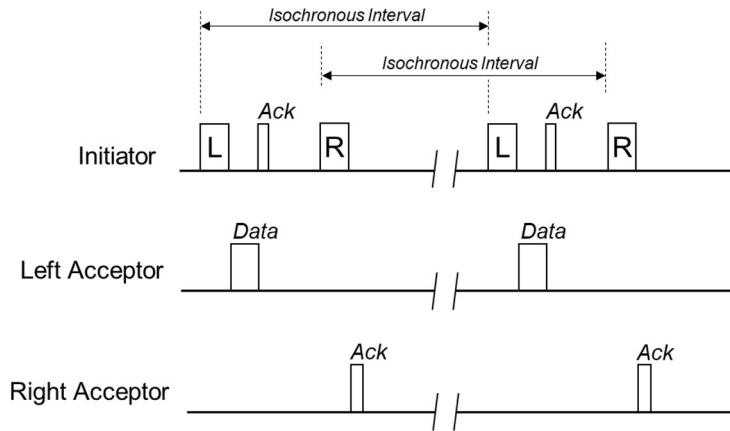


Figure 4.19 Example of a bidirectional CIS for the left earbud and a unidirectional CIS for a right earbud

In the example shown in Figure 4.19, the Initiator has set up individual CISes with a left Acceptor and a right Acceptor. Both receive data from the Initiator, but the left one also sends data from its microphone back to the phone. The same principles we've seen before apply. Data is sent by the Initiator, the Acceptor immediately responds to acknowledge receipt (or not) and if it has data, includes it in the return CIS PDU. If that return PDU gets back with a good CRC, the Initiator will acknowledge it, close the event, and transmit data to the right Acceptor CIS.

The acknowledgements in both directions use the NESN and SN bits in the ISO PDU header, flipping the bits when a packet is acknowledged. In Figure 4.19, all of the transmitted packets have the same ISO PDU format. Those marked “Ack” contain no audio data, so would have the NPI bit set to one to indicate they have a null CIS PDU. The data packets from the left Acceptor will be an identical format to the Initiator’s “L” and “R” packets, but will contain data from the Acceptor’s microphone.

The value of NSE applies to both directions in a CIS, as the same Subevents are used for transporting PDUs both to and from the Acceptor. Once the payload in one direction has been acknowledged, future transmissions can replace the payload of that PDU with a null payload. (That has no effect on the timing of the Subevents, but saves a small amount of power.)

The Initiator can set the CIE bit in the ISO PDU header in a uni- or bidirectional CIS to indicate that it is not going to transmit any further payload data in the current CIS Event. It should not close the CIS Event unless the payloads for both directions have been acknowledged. If an Initiator has had its packet acknowledged, but has not received an expected packet from an Acceptor, it should continue to transmit CIS Null PDU packets in every available Subevent, to give the Acceptor opportunities to retransmit.

An Acceptor can also set the CIE bit in its ACK to indicate that it will not respond to any further packets in that CIS Event.

Section 4.3 - Connected Isochronous Streams

The two directions in a bidirectional CIS can have different properties, such as codec and PHY and may even be used by different applications. Even some of the structural parameters can be different for the two directions. FT and BN can be different, as can the Max_PDU and Max_SDU values, which also means that the SDU_Intervals can be different, although one needs to be an integer multiple of the other. However, the ISO_Interval, Packing, Framing and NSE must be the same.

4.3.6 Synchronization in a CIS

Once we add a second Acceptor, both Acceptors act independently of each other, but under the Initiator's control. If they are a Coordinated Set (which we covered in Section 3.9), they will know the other one exists, because they know how many members there are in the Coordinated Set. However, your left ear bud and right ear bud don't necessarily know the other one is present, turned on or receiving data. A user may take one out to share with a friend, or the battery in one may die. Even if they have a means of communicating, there is no guarantee that they will always be in contact with each other. To cope with this and enable them to render or capture audio streams at precisely the same time, the Core's Isochronous Channels feature includes a synchronization method which allows microsecond level accuracy of rendering between two or more Acceptors, without any Acceptor needing to have any knowledge of the presence (or otherwise) of any other Acceptors. In case you're doubting it, this is microseconds, not milliseconds. Figure 4.20 illustrates how this is done.

Within a CIG, CIS Events are scheduled for each of the CISes in that CIG. For a pair of earbuds that will be two CIS Events - one for the left earbud one for the right earbud. It could be more, such as with multi-channel sound systems. These CIS events make up a CIG event. In Figure 4.20, for the sake of simplicity, the multiple CISes are shown as sequential. They could equally be interleaved.

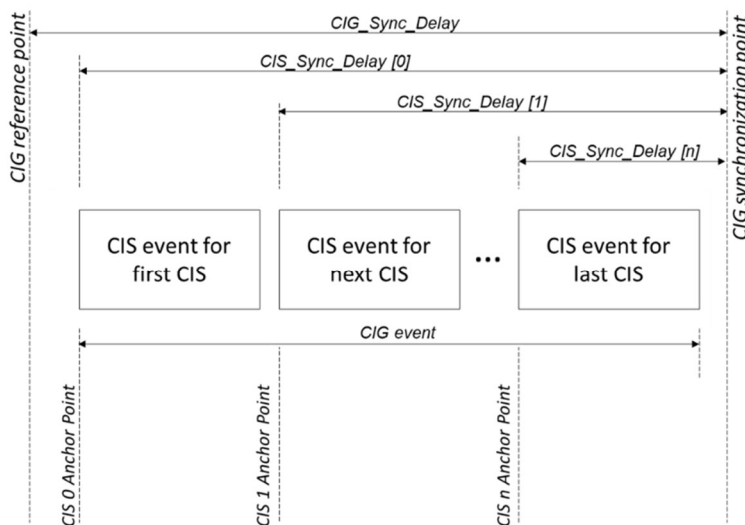


Figure 4.20 Synchronization of multiple CISes

The Core defines a CIG reference point which may be coincident with, but cannot be later than the Anchor Point of the first CIS. Typically, it will occur slightly before that. It also defines a CIG synchronization point which occurs after the last possible receive event for the last CIS event within that CIG. At that point, the Initiator knows that every Acceptor will have had every possible opportunity to receive every PDU that has been sent to it and time to return any data for the Initiator.

In order to reconstruct this common point, every device is informed of its individual CIS Sync Delay for every CIS which it supports, which is calculated from the Instant³³ for the ACL link associated with that CIS. This allows it to calculate the CIG synchronization point, which is a common timestamp across every device. With this knowledge, each device can determine when it should start to decode the audio. BAP adds a feature called Presentation Delay, which tells the Acceptor when to render the decoded stream. As we move up the layers and start to dig into the detail of the basic concepts of QoS and latency, we'll see how Presentation Delay is used to add flexibility to the rendering time.

If you have devices with microphones, the same synchronization problem exists, but in this case you need to coordinate the point at which audio is captured by an Acceptor. If it uses the uplink of a bidirectional CIS, it will use the same CIG synchronization point, but is likely to require a different value for Presentation Delay to the one used to render the downlink audio data from the Initiator, as the Acceptor needs extra time to capture and encode the audio stream.

The synchronization timing is defined with a microsecond accuracy. This means that the Controllers of all of the Acceptors within a CIG have a timestamp assigned which will be within a few microseconds of each other. That needs to be conveyed to the application layer that renders the audio streams. The method of doing that is down to the implementation. The specifications do not define an accuracy for the final rendering of audio data, but typical implementations achieve values of between 5 to 10µsecs for left to right synchronization, which means that left and right earbuds can present the two audio streams to the ears closer together than the brain can discern.

4.3.7 The CIG state machine

Having covered all of the features that make up Connected Isochronous Streams, we can look at how to put them together to configure and establish a Connected Isochronous Group.

³³ An Instant is a timing reference, normally the start of a transmitted packet, which is used for synchronization.

Section 4.3 - Connected Isochronous Streams

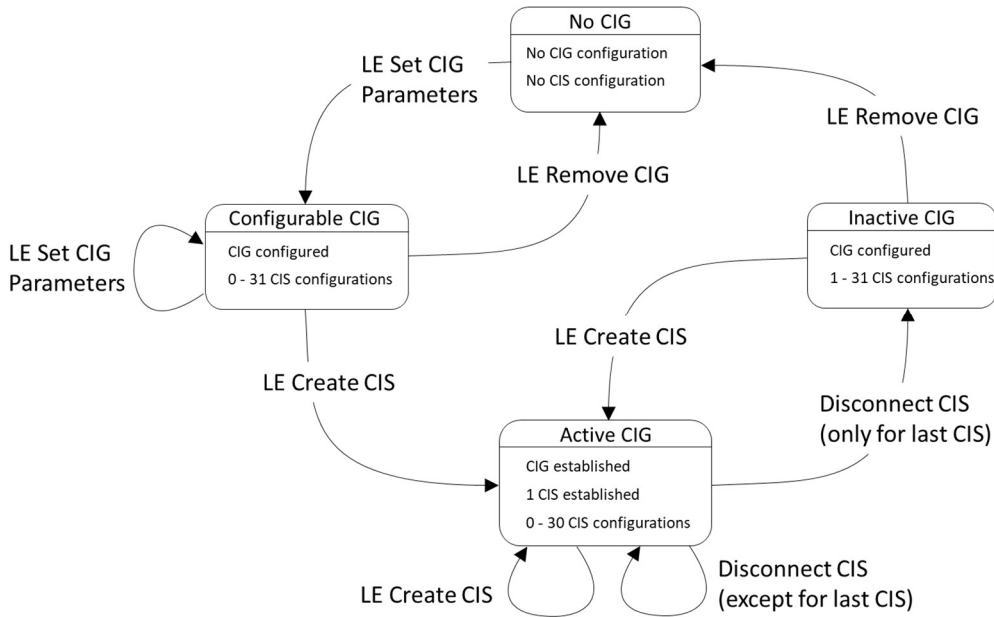


Figure 4.21 The CIG state machine

There's a fairly straightforward state machine used to configure and establish a CIG and its constituent CISes, which is shown in Figure 4.21. It contains three states:

- The Configurable CIG state, which is where all of the CISes that make up a CIG are defined. That's done through a set of HCI commands³⁴ called the LE Set CIG Parameters command. Once these are sent to the Controller, the Initiator has all the information that it needs to work out the scheduling and to optimise its airtime usage. The parameters of a CIS can be modified within the Configurable CIG State.
- The Active CIG state, in which one or more of the configured CISes is enabled. The CIG moves to the active state by enabling at least one of its configured CISes using the LE Create CIS HCI command. It doesn't need to enable all of the configured CISes. Once the CIG is in the Active state, it can disconnect individual CISes, and it can use the LE Create CIS command to enable other CISes which it has previously configured. In theory, that allows it to swap CISes whilst that CIG is active. This can be useful if there is only an occasional need for a CIS, such as a return path for voice commands. The CIS could be turned on and off depending

³⁴ HCI (Host Controller Interface) commands are defined to provide the link between the Host stack and the Controller. They are used for prototype testing and provide an interoperable interface where the Host stack and Controller silicon come from different manufacturers. Generally, operating systems do not make them available, providing a higher level API for developers.

on when it is needed. Once a CIG is in the active state, you cannot configure additional CISEs. That can only be done by disconnecting all CISEs, deactivating the CIG and starting again.

- The Inactive CIG state. A CIG moves to the Inactive state by disconnecting all of its established CISEs. It cannot change the configuration of any CIS in this state, nor add new ones, but it can return to the Active state by using the LE Create CIS HCI command to re-establish a CIS. This allows a CIG to be reactivated without the need for the Controller to go back to the start and reschedule one or more of its configured CISEs.

When all of the established CISEs have been disconnected, the CIG can be removed by issuing the LE Remove CIG HCI command.

4.3.8 HCI commands for CISEs

The Core specification defines five HCI commands which are used to inform the Controller about each CIG. The first is the LE Set CIG Parameters command [Vol 4, Part E, 7.8.97], which creates a CIG and configures an array of CISEs within it. Each CIS within the CIG can be unidirectional or bidirectional. Each one can use a different PHY for each direction.

The first use of the LE Set CIG Parameters command creates the CIG, moving it to the Configured CIG state, with the number of CISEs defined in the array. The command can be used multiple times, to add more CISEs to the CIG, or modify CISEs which have already been defined. Each CIS is assigned a Connection handle.

Each time the command is issued, the Controller should attempt to calculate a schedule for all of the CISEs in the CIG, which meets the requirements specified by the HCI parameters, and, if successful, will confirm this with an HCI Complete Command. Note that two of the parameters sent by the Host using the HCI command – Packing and RTN (the number of retransmissions), are only recommendations. The Controller should try to accommodate them when working out the schedule, but can ignore them, or attempt a best-case schedule that is guided by them. The Link Layer parameters of Burst Number, Flush Timeout and NSE are determined by the Controller's scheduling algorithm and cannot be explicitly set by a higher layer specification. Some of the profile specifications provide recommendations that suggest optimal settings for specific use cases, but it is up to a specific chip implementation as to whether these are accessible by an application, or determined solely by the scheduler.

This mismatch between the HCI parameters sent from the Host and those set by the Controller may seem odd, but there is a good reason, which is to prevent a top level profile or application from trying to prioritise the overall radio operation. Many Bluetooth chips include Wi-Fi, and they typically share an antenna. Hence the Controller needs to work out how to fit in the demands of both. The Bluetooth technology implementation may also be running multiple profiles. There is no reason a smart phone can't receive a Bluetooth LE Audio broadcast stream and then retransmit it to a pair of hearing aids as a pair of Connected

Section 4.4 - Broadcast Isochronous Streams

Isochronous Streams (other than the physical constraints of available airtime and processing resources). However, at a profile level, none of these applications may know that the other exists. If each one could dictate the exact values of BN, FT and NSE, it would be very likely that they would be unable to coexist with the constraints of other applications. That's why the HCI commands prevent this level of control, allowing the Controller to work out how best to fit everything together. The scheduling algorithms are not accessible to application developers – they are a critical part of the Bluetooth chip. However, it's important to understand why you can't dictate what you want and to be aware of the dangers of over-specifying the needs of your audio application.

Once all of the CISes have been configured, the LE Create CIS command is used to create each CIS which is required, which causes the CIG to move to the Active CIG state. Not all CISes need to be created at this point. As long as one of them is created using the LW Create CIS command, the CIG will also be established. From this point onwards, a CIS can be disconnected, and another CIS created at any point, which may be useful when an application decides to move from using a microphone in an earbud to the microphone on a smartphone. However, this does assume that the Controller managed to schedule all of the CISes and may result in airtime being allocated that is then unavailable for other uses.

The CIG remains in the Active CIG state until the last CIS is disconnected. At this point it moves to the Inactive CIG state. It can either be permanently removed with the LE Remove CIG command, or returned to the Active CIG state by using the LE Create Command again on at least one of the CISes.

As each CIS is created by the Initiator, each Acceptor will be informed of the connection parameters via Link Layer commands, leading to an HCI LE CIS Request event [Vol 4, Part E, 7.7.65.26] being sent to its Host. If it accepts the request, it will respond with an HCI LE Accept CIS Request Command [Vol 4, Part E, 7.8.101], leading to both Initiator and Acceptor Hosts receiving an HCI LE CIS Established event [Vol 4, Part E, 7.7.65.25]. When we get to the ASCS, we'll see how these work with the Isochronous Stream state machines.

That completes a review of all of the component parts of a CIG and CIS and how we put unicast Connected Isochronous Streams together. Now, we'll look at the analogues for broadcast where we have to do the same thing, but without the luxury of having a connection between the two devices to allow them to negotiate what they're doing.

4.4 Broadcast Isochronous Streams

Broadcast audio is a totally new concept for Bluetooth technology. In the past, audio has always been streamed directly from one device to another device, as we saw in the previous chapters. That concept has now been extended with Connected Isochronous Streams, so that we can stream audio to multiple devices. But those devices are still connected and every packet

that is correctly received is acknowledged. With broadcast, there is no connection³⁵. Any device which is within range of a Broadcast Transmitter can pick up the broadcast Audio Streams. The big difference from Bluetooth Classic Audio profiles is that it is one-way – there are no acknowledgements. That means that it is technically possible to build a Broadcast Source which does not contain a receiver, but which only transmits. Where Broadcast Streams are encrypted, the receiving device will need the appropriate Broadcast_Code to decrypt them.

There is a practical advantage in having no acknowledgements, which is that it generally results in a much greater range. That occurs because the link budget over a connection is often very asymmetric. Broadcast Transmitters are frequently mains powered, so can easily transmit at the maximum allowed power. That gives it a good link budget to an earbud. However, an earbud is unlikely to be transmitting acknowledgements back at anything more than 0dBm (1mW), both to conserve its battery and because its small antenna is probably going to have a gain below 0dB. That means that the link budget for the earbud to receive a broadcast transmission may be 10 – 20dB higher than for the return acknowledgement path. It is the latter which determines the maximum range for a CIS. Without the constraint of receiving weak acknowledgements, a Broadcast Transmitter can cover a considerable area – far more than can be achieved with a Connected Isochronous Stream transmission, or with classic BR/EDR implementations. Demonstrations of Broadcast Transmitters running at 4mW have achieved usable ranges in excess of 50 metres.

4.4.1 The BIS structure

The structure of Broadcast Isochronous Streams and Groups is very similar to what we've just looked at with Connected Isochronous Streams, but with the big difference that it is one way and not acknowledged.

Figure 4.22 shows that Broadcast Isochronous Streams have the same basic PDU structure of header, payload and an optional MIC if you want encryption. However, the header for the BIS is a lot simpler, as it does not need the SN and NESN flow control bits that are in the CIS PDU header. It starts with the same two Link Layer ID (LLID) bits. Those indicate whether the PDU is framed or unframed. Then come CSSN and CSTF, which are used to signal the presence of an optional Control Subevent. CSSN is the Control Subevent Sequence Number and CSTF is the Control Subevent Transmission Flag, signalling that a Control Subevent is present in that BIS Event. These are new and don't exist in Connected Isochronous Streams. Within a BIG, there is a Control Subevent which can be used to provide control information to every Acceptor. We need these in broadcast, as there's no ACL to inform Acceptors of things like a change in the hopping sequence. The only other information in the header is the

³⁵ As we will see later on, there can, and often will be, a connection. But that enhances the way that broadcast works – it doesn't affect the structure of a BIS or BIG, so we'll ignore it for the time being.

Section 4.4 - Broadcast Isochronous Streams

length of the payload.

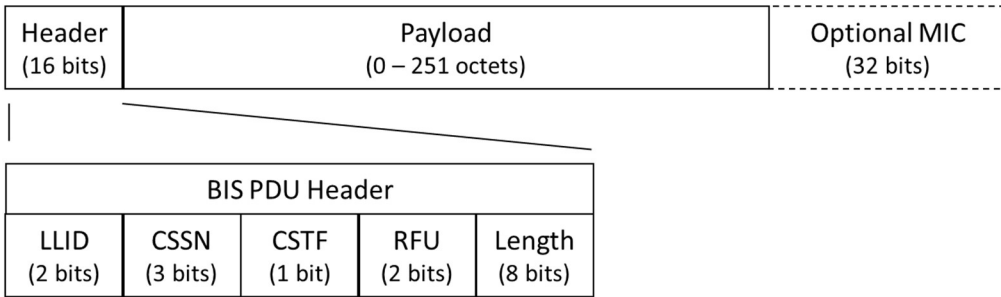


Figure 4.22 Isochronous PDU and header for Broadcast

If we look at the structure of a Broadcast Isochronous Stream in Figure 4.23, it's also very familiar. The fundamental difference between a BIS and a CIS is that there are no acknowledgments within a BIS. A BIS is composed purely of Subevents which contain data, sent by the Initiator. The data flow is not bidirectional – everything is transmitted from the Broadcast Source. As with CISes, each Subevent is transmitted on a different frequency channel.

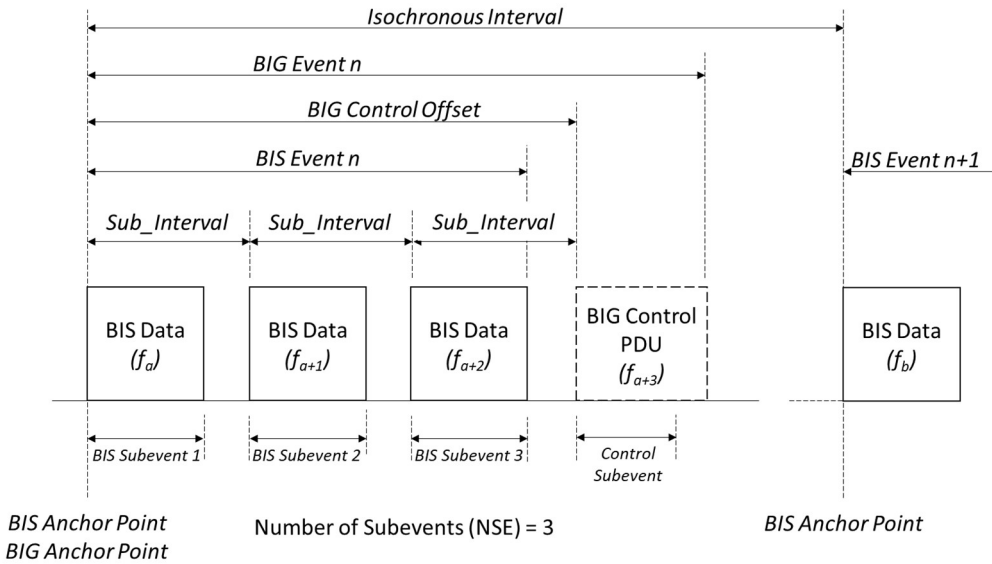


Figure 4.23 Structure of a single Broadcast Isochronous Stream

A notable difference between a BIG (a Broadcast Isochronous Group, made up of one or more BISes) and a CIG, is the potential existence of a Control Subevent, which is shown dotted in Figure 4.23. When it occurs, (which is in BIG Events where the header of the ISO PDUs contains the CSTF flag), the Control Subevent is transmitted after the final Subevent of the last BIS, and provides an opportunity for control information to be sent to every device

which is receiving a broadcast stream. We'll cover what it does in Section 4.4.3.

As Figure 4.23 shows, a Broadcast Isochronous Stream has the same basic elements of an Isochronous Interval, and Anchor Points for BIS and BIG Events. As there is no acknowledgement or return packet, a Sub_Interval time is defined, which is the time between the start of consecutive Subevents within a single BIS. It is also the time between the start of the final Subevent of the last BIS and a Control Subevent, if one is present.

If the BIG contains more than one BIS, each BIS is separated by a BIS_Spacing. By adjusting the values of the Sub_Interval and the BIS_Spacing, the BISes can be arranged in either a Sequential or Interleaved fashion, which is illustrated in Figure 4.24 and Figure 4.25. If the BIS_Spacing is bigger than the Sub_Interval, they will be arranged sequentially. If it's smaller, they will be interleaved.

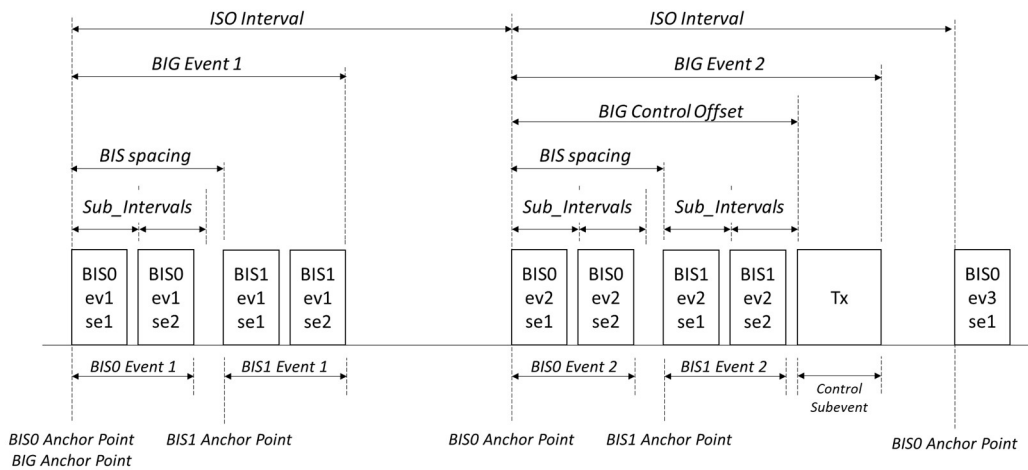


Figure 4.24 Sequential BISes

Both diagrams show two BISes, each of which have NSE set to two (i.e., two Subevents each). There are some important observations that can be made from these figures. The first is that the Control Subevent is never counted in the NSE – it is a totally separate Subevent. The second is that when the Control Subevent is present it does not affect any of the other packets or the Anchor Points. It is scheduled immediately after the last Subevent of the last BIS. But it does extend the BIG Event for the Isochronous Intervals which contain a Control Subevent.

Section 4.4 - Broadcast Isochronous Streams

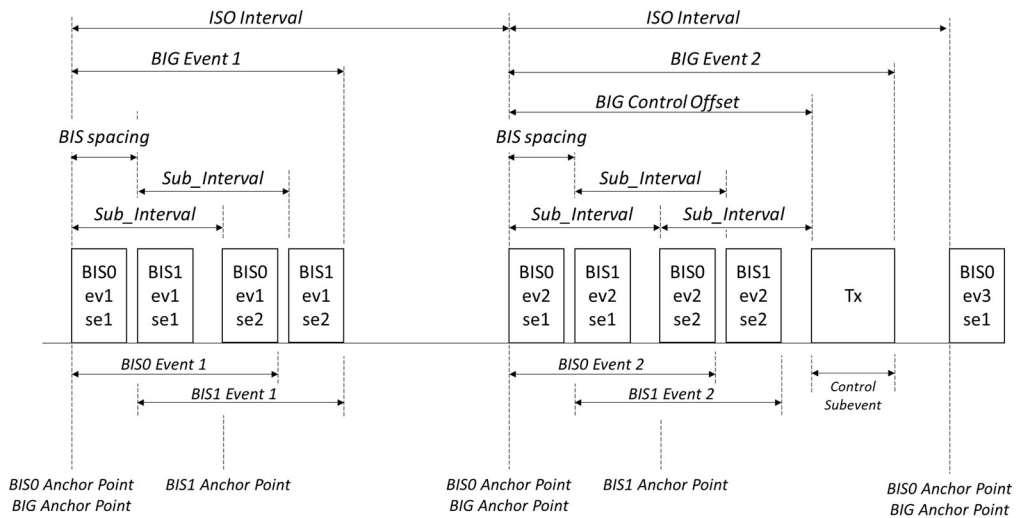


Figure 4.25 Interleaved BISes

Whereas a CIS can stop transmitting audio packets once a device has received them, a Broadcast Source has no idea whether or not they have been received, so it has to repeat every transmission. However, as soon as an Acceptor has received a packet, it can turn its radio off and wait for the next scheduled BIS Event. This again highlights the asymmetry that we have within Bluetooth LE. Because the Broadcast Source is always transmitting, it typically has a much higher power drain than the Acceptors, which are likely to be earbuds. In general, that means that Broadcast Sources need bigger batteries, or a permanent power source. As they are typically phones or infrastructure transmitters in public places, that's not generally an issue. However, it should be kept in mind if broadcast transmission is being embedded into small devices like watches or wristbands.

Unlike CISes, all BISes have the same timing structure. Whereas the structure of each individual CISes is normally tailored to its codec configuration, optimising the Subevents for the PDU size, every BIS Subevent is the same length, which must fit the largest PDU that is being transmitted³⁶. That constraint is because the overall BIS timing structure is defined in the BIGInfo, which is included in the Periodic Advertisements. The BIGInfo structure is limited in size, so doesn't have the granularity to specify different timings for different BISes – it specifies a “one size fits all”. It means that if you want to broadcast one channel at a 48kHz sampling rate and a second one at 24kHz, the smaller 24kHz channel would still be allocated BIS Subevent timings for the larger 48kHz packets, which wastes airtime. If that causes a problem, an alternative is to transmit the packets in separate BIGs. In this case, that would mean one BIG for the 48kHz sampled packets and another for the 24kHz sampled

³⁶ For low sampling rate applications, it is possible that the largest PDU could be the Control Subevent, which would then determine the size of every BIS subevent in the BIG.

ones. The downside is that this adds complexity and doubles the amount of advertising, as each needs its own advertising set. Implementers need to decide which works best for their specific application. Designers need to consider whether to place all of the BISes into a single BIG. In some applications it may be a more efficient use of airtime to separate them into different BIGs.

4.4.2 Robustness in a BIS

The way the parameters of a BIS are defined is a little different to a CIS, as without acknowledgments, we need to employ some different strategies to maximise the chance of a transmission being received.

A BIS still has a Burst Number (BN), which is the number of payloads within each Isochronous Interval, and a Number of Subevents (NSE), defined in the same way as that for CISes. To compensate for the lack of acknowledgements, the Core introduces the concept of a Group Count, which introduces an additional level of diversity into the way that retransmissions are sent.

Group Count is defined as the ratio of NSE to Burst Number (NSE/BN) within an Isochronous Interval. It is used to determine the way that retransmissions are arranged in a BIS, allocating them to Groups, which are used to add diversity into the transmission scheme. (These Groups have nothing to do with Broadcast Isochronous Groups – it is an unfortunate use of the same word for two totally different concepts.)

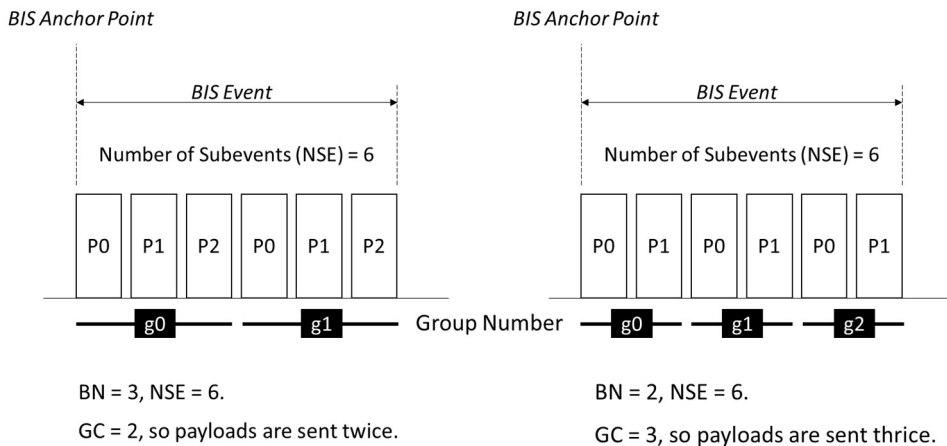


Figure 4.26 The effect of Group Count and the assignment of Group Number

Figure 4.26 shows two examples of a BIS, each of which contains six Subevents, so $NSE = 6$. In the first case, we have set a Burst Number of 3. That means that each of the three payloads which are available for that BIS event are sent twice. In the second example, on the right, there are the same number of Subevents, but with a Burst Number of 2, so we have three retransmissions of the two payloads. Each individual Subevent is given a group number (g) that goes from 0 up to (Group Count - 1). So, in the left-hand diagram, we have a Group Count

Section 4.4 - Broadcast Isochronous Streams

of 2, comprising group 0 and group 1. In the right-hand example, there are three groups: group 0, group 1 and group 2, with a Group Count of 3.

An important thing to note is that although these two examples look the same, they will have different Isochronous Intervals. Assuming that our payloads each represent a codec frame containing 10ms of encoded audio, the example on the left will have an Isochronous Interval of 30ms, as it contains three payloads – P0, P1 and P1. The example on the right will have an Isochronous Interval of 20ms, as there are two payloads, P0 and P1. It illustrates the effect Burst Number has on latency, as an Acceptor has to wait longer before it can render audio.

Group Count works with two other, new parameters, which are included in the Core specification to increase the temporal diversity of transmissions, offering enhanced robustness:

- Immediate Repetition Count (IRC), which defines the number of Groups which carry data associated with the current event, and
- Pre-Transmission Offset (PTO). The concept of Pre-transmission is to allow “early”³⁷ transmission of packets in the hope that a receiving device can receive audio data packets early, allowing it to power down, and then be ready to render them at the point that the final transmission opportunity occurs.

These parameters, which are calculated by the scheduler in the Controller, replace the Flush Timeout that is used within a CIS. For a CIS, Flush Timeout is essentially an end-stop, terminating transmissions of a PDU. In most cases, it’s never needed, because as soon as an Initiator receives an acknowledgment that its data has been received, it can close the event and stop transmitting. A Broadcast Source can’t do that – it has to keep on transmitting. Therefore, it’s advantageous to maximise the diversity of transmissions of packets to give every Acceptor the best chance of finding a packet. The sooner they can do that, the sooner they can go to sleep until the next one arrives.

Pre-Transmission Offset works by introducing the concept of Subevents associated with a current event as well as Subevents associated with future events. This allocation of Subevents to PDUs is even more complex than the scheme for CIS, so the best was to explain it is to go through a series of examples showing the effect of changing the values. This is all worked out by the Controller and is not accessible to an application, but it helps to have an understanding of it when you set the HCI parameters to configure your streams.

To illustrate the concept of future Subevents, Figure 4.27, demonstrates a BIS event where we have an NSE of eight Subevents; the first five of which are associated with the current BIS

³⁷ It isn’t really early – it’s just stretching the transmission out over more Isochronous Intervals, increasing the latency. But for some historic reason, this technique uses the word “early”.

event, the last three of which are used for data from future BIS events.

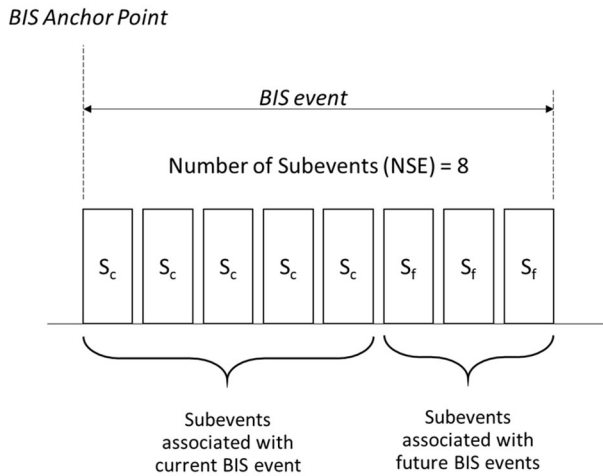


Figure 4.27 The concept of future BIS events

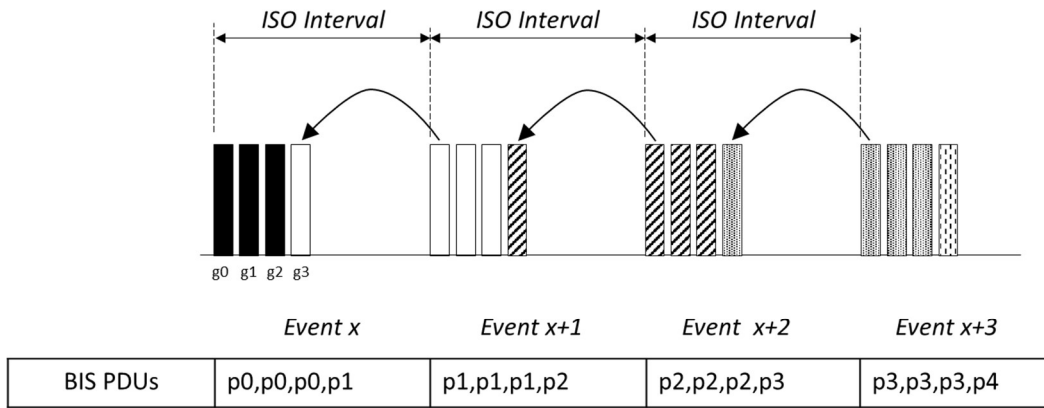
We can now put everything together with some more examples.

These groups of subevents are given a number (g), which we first saw in Figure 4.26. This, along with the Immediate Repetition Count (IRC), determines which payloads are sent in each transmission slot according to the following rules, which are defined in Core Vol 6, Part B, 4.4.6.6:

- If $g < \text{IRC}$, group g shall contain the data associated with the current BIS event.
- If $g \geq \text{IRC}$, group g shall contain the data associated with the future BIS event that is $\text{PTO} \times (g - \text{IRC} + 1)$ BIS events after the current BIS event.

We'll now look at a few examples which illustrate how these different parameters result in a variety of different retransmission schemes.

Section 4.4 - Broadcast Isochronous Streams



NSE = 4, BN = 1, IRC = 3, PTO = 1. (GC = 4)

In Event x:

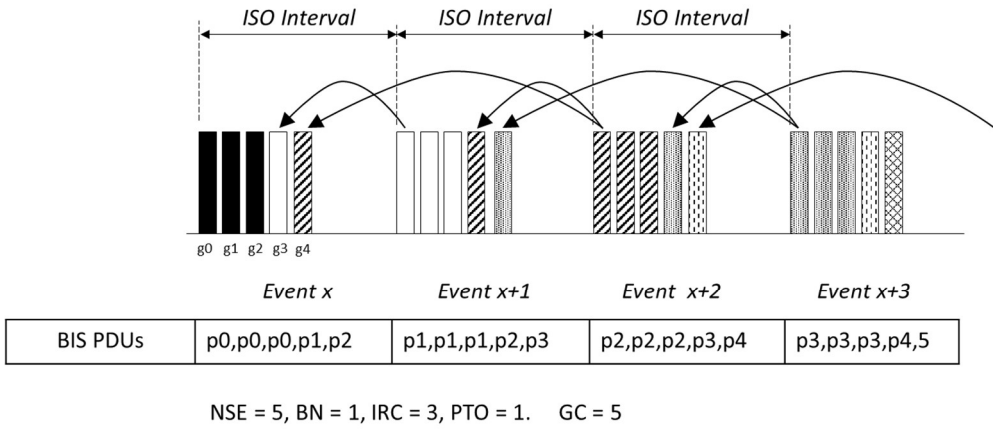
g0, g1 and g2 use data associated with Event x

g3 uses data associated with Event x+1

Figure 4.28 Pretransmissions with NSE=4, BN=1, IRC=3 and PTO=1

In Figure 4.28, we have four Subevents with an NSE of 4. The Burst Number of 1 means one new payload is provided within each BIS event (so the Isochronous Interval for this example is 10ms). The Immediate Repeat Count is three, which means that the data associated with each event is transmitted in the first three slots. With the values of IRC = 3 and PTO = 1, the last transmission in that BIS event comes from the next BIS event. Therefore, within every BIS event there are always three transmissions of the current data plus one transmission of the data from the next event.

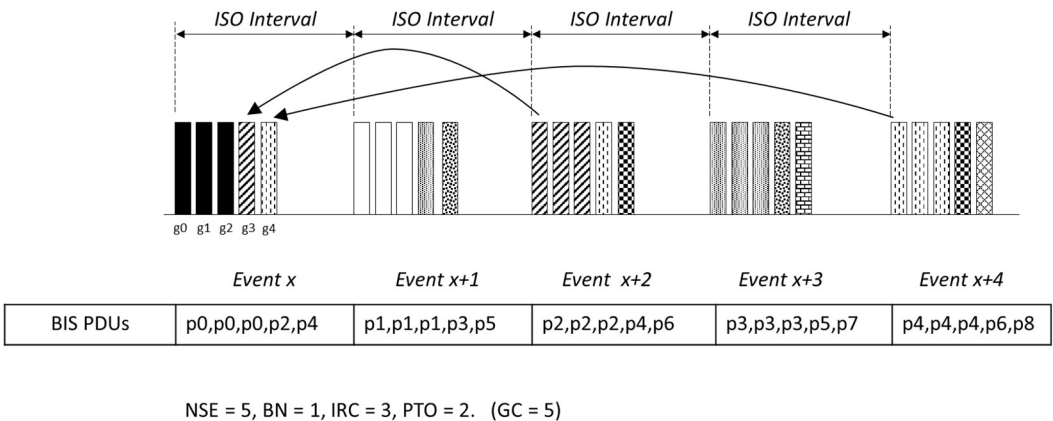
In case it seems like we've invented time travel, we haven't. We've just bent our definitions slightly. Event x can't happen until we have both the p0 and p1 PDUs available. So p1 is really the current data in Event x. This demonstrates that as soon as PTO is greater than 0, the latency starts to increase, as the Sync reference point cannot occur until after the last possible transmission of data, which for p1 is in Event x+1.



In Event x:
 g0, g1 and g2 use data associated with Event x
 g3 uses data associated with Event x+1
 g4 uses data associated with Event x+2

Figure 4.29 Pretransmissions with NSE=5, BN=1, IRC=3 and PTO=1

Figure 4.29 shows what happens when we increase the number of Subevents to 5. Here, the IRC remains at 3 but as NSE is 5, that provides two Subevents which are associated with data from future BIS events. These are used to pre-transmit a packet from event x+1 and event x+2. That means that transmissions are being spread across more Isochronous Intervals, with data coming from three consecutive BIS Events. That helps to provide robustness against bursts of wide-band interference which may last more than one Isochronous Interval, but it is at the expense of greater latency, which has grown by another 10ms.

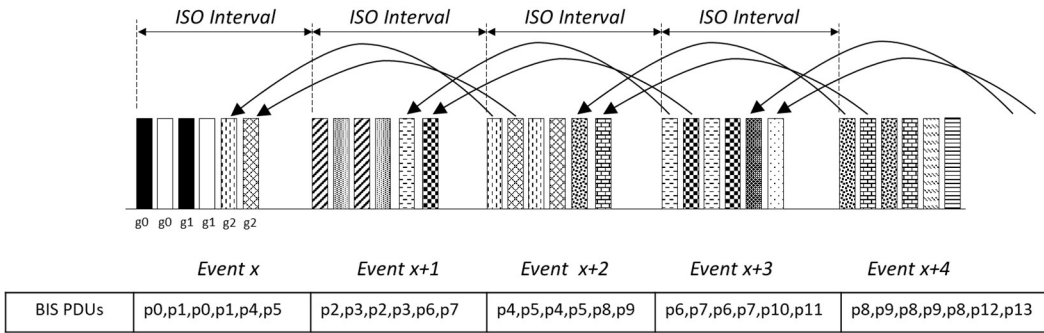


In Event x:
 g0, g1 and g2 use data associated with Event x
 g3 uses data associated with Event x+2
 g4 uses data associated with Event x+4

Figure 4.30 Pretransmissions with NSE=5, BN=1, IRC=3 and PTO=2 (only event x arrows shown)

Section 4.4 - Broadcast Isochronous Streams

Figure 4.30 looks at the case where we are spreading even further, by increasing the Pre-Transmission Offset to 2. This results in the inclusion of data from an $x+2$ BIS event and an $x+4$ BIS event, effectively spreading the audio data transmissions over five events and adding a total of 40ms to the latency compared with what it would be with $PTO = 0$. In this figure, we're only showing the arrows for group 3 and group 4 in that first BIS Event x , otherwise the figure becomes very cluttered and difficult to read.



$NSE = 6, BN = 2, IRC = 2, PTO = 2. (GC = 3)$

In Event x :

$g0$ uses data associated with Event x

$g1$ uses data associated with Event x

$g2$ uses data associated with Event $x+2$

Figure 4.31 Pretransmissions with $NSE=6, BN=2, IRC=2$ and $PTO=2$

Finally, in Figure 4.31, we've set NSE to 6, Burst Number and IRC to 2, and the Pre-Transmission Offset is also 2. With these settings, each BIS event includes the two packets for that "current" event transmitted twice, one after the other, as we saw in Figure 4.26, with the final two Subevents used for packets two events in the future. Because BN is 2, the Isochronous Interval will be 20ms. As the PTO is 2, $p4$ and $p5$ come from two Isochronous Intervals in the future, so the latency is 50ms greater than the simple example of Figure 4.28.

These BIS parameters allow for some very flexible transmission schemes to cope with different requirements in terms of latency and robustness. However, these are Core parameters, which are set by the scheduler in the chip's Controller. When configuring a BIG (in which all BISes have the same basic Isochronous parameters), a host application can only request the number of retransmissions (RTN , which is $NSE - 1$), the SDU interval and the Maximum Transport Latency. As with CISes, the retransmission number is only a recommendation. It is up to the chip to decide how to translate those parameters into values for NSE, BN, IRC and PTO . This means that the same host application running on different silicon could result in different configurations. If a Broadcast Transmitter is concurrently running other Bluetooth and Wi-Fi applications, as with a smartphone or PC, it could also mean that the configuration may change from session to session.

In the next chapter, where we discuss Quality of Service, we'll see how they can be used for different broadcast situations. However, these parameters are largely inaccessible to an applications developer, unless a chip vendor provides a proprietary means to access them.

In conclusion, both Flush Timeout and Pre-Transmission Offset increase latency. For a CIS, Flush Timeout helps to share battery savings between both Initiator and Acceptor, because the use of acknowledgements means that events can be closed. With a BIS, where there are no acknowledgements, a Broadcast Transmitter has to transmit at every Subevent. PTO effectively gives all of the power savings to the Acceptor, by providing a diversity of transmission that gives it the best chance of acquiring a packet. However, these savings need to be balanced by the impact on latency. Note that this balance is determined by the Initiator, so designers of Broadcast Transmitters need to consider the impact of their choices on the Acceptor, as they may have an effect on their battery life. Equally, designers of Acceptors need to be aware to the range of schemes that Broadcast Transmitters may throw at them.

4.4.3 The Control Subevent

The Control Subevent [Core, Vol 6, B, 4.4.6.7] is generally a rare event and does not need to be included within every BIG event. Typically, Control Subevents are used for items such as channel map updates, so only occur occasionally, when that information needs to be provided to all of the devices that are receiving the broadcast Audio Streams. The advantage of using Control Subevents is that devices no longer need to scan to find out the basic BIG information, such as the hopping channels that are being used. This minimises the amount of work that a receiver has to do to ensure that it stays synchronized with a BIG and the BISEs within it. Without them, a Broadcast Sink would need to continue to receive the Periodic Advertising train and regularly examine the BIGInfo to discover any changes. This would almost certainly result in synchronization being lost for a time at the point the hopping sequence changed. A Broadcast Sink could constantly monitor the BIGInfo to look for changes, but that is undesirable, because of the battery drain. Hence the reason for the Control Subevent.

When a Control Subevent is used, it is transmitted in six consecutive BIG events. It may then be transmitted in any subsequent BIG event, but only one control event can be transmitted at a time – you are not allowed to interleave different Control Subevents. Every BIS header for a BIS Subevent in a BIG event which includes a Control Subevent must have the Control Subevent Transmission Flag (CSTF) set to 1 to signal the presence of a Control Subevent in that BIG event. Its Control Subevent Sequence Number (CSSN) will tell a receiver if it is the same as one which they have already received. It increments by 1 (with 7 wrapping to 0) at the start of a BIG event containing a new Control PDU. Control events use the same frequency hopping scheme as every other Subevent, taking the index of the first BIS event in the same BIG event.

At the moment, the only use defined for the Control Subevent is for updates to the hopping sequence of the broadcast transmissions. Broadcast Transmitters which intend to update their

Section 4.4 - Broadcast Isochronous Streams

hopping sequence will need to use it, otherwise Acceptors will lose synchronization when the hopping sequence changes and will need to resynchronize, resulting in a break in reception. An Acceptor can work out whether a Broadcast Transmitter is supporting Control Subevents by calculating whether the `BIG_Sync_Delay` value includes the extra BIS event required by the occasional Control Subevent. As the Control Subevent has a variable length and is usually short, it generally has a minor impact on airtime usage.

4.4.4 BIG synchronization

As with Connected Isochronous Streams, individual receiving devices, typically a pair of earbuds or hearing aids, don't necessarily know about each other's existence. To keep their audio rendering in synchronization they need to use information that's included within the BIG to understand when they need to render their decoded audio packets. To enable them to do this, a BIG Synchronization Point is defined, which coincides with the end of the transmission of the audio data. Because we have no acknowledgments to take into account in broadcast, that BIG Synchronization Point is located exactly at the end of the last BIS transmission of the last BIS in a BIG. Normally, that will be coincident with the end of the BIG event. However, for the case when there is a Control Subevent present, the BIG Synchronization Point remains at the end of the last BIS Subevent transmission, whilst the BIG event extends to the end of the Control Subevent, as shown in Figure 4.32.

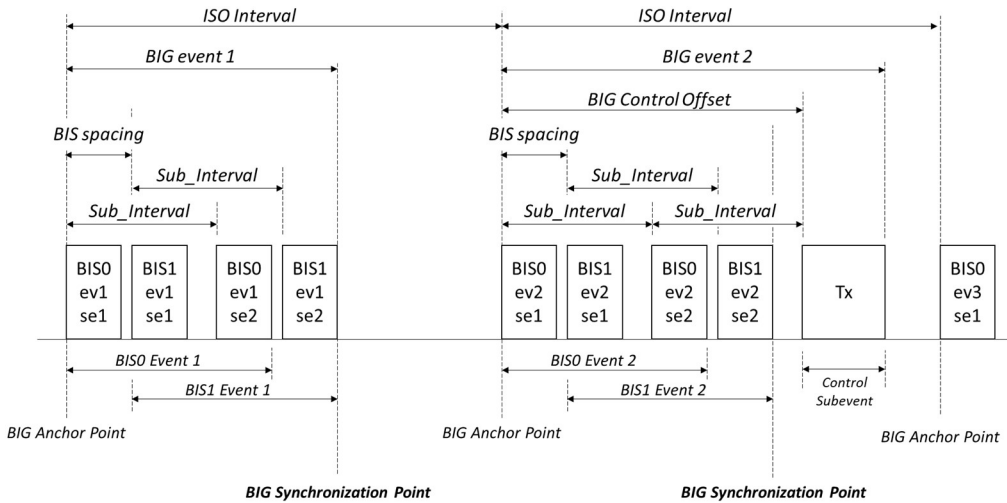


Figure 4.32 BIG synchronization

At the BIG Synchronization Point, every device that is listening to any of the Broadcast Isochronous Streams within that BIG will know that every other device has received its data; hence the BIG Synchronization Point is a fixed point in time at which they can each apply the Presentation Delay. This is defined higher up the stack and dictates the point at which audio needs to be rendered. The important point here is that audio is not rendered at the BIG Synchronization Point – it's rendered at the end of the Presentation Delay, which commences at the BIG Synchronization Point. As broadcasting consists only of transmissions from a

Broadcast Source, the Presentation Delay value only applies to rendering at a Broadcast Sink.

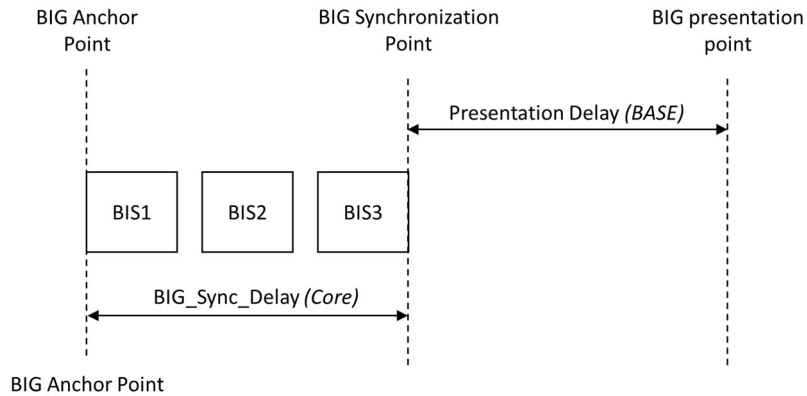


Figure 4.33 The application of Presentation Delay in a BIG

The derivation of the BIG Synchronization Point is also a little different for broadcast. Unlike unicast, every BIS has the same basic timing parameters – that’s because they need to be defined in the BIGInfo structure, and there is not enough space in the packet which carries it to allow different settings for individual BISes. This means that the BIG Synchronization Point is a fixed interval from the BIG Anchor Point, both of which are known by every Acceptor, as that information is contained in the BIGInfo. (In contrast, each CIS uses a CIS_Sync_Delay based on its own Anchor Point, as it doesn’t know the relative timings of any other CISes and can’t assume they are the same as its own).

Figure 4.33 shows the example where $PTO = 0$. If retransmissions occur over multiple BIS events, then the application of Presentation Delay needs to start at the BIG_Sync_Delay datum associated with the final BIS_Event for that packet. Note that if Control Subevents are supported, they will occur immediately after BIG_Sync_Delay, so controllers need to be aware that they will be processing them at the same time as they are decoding the received codec packets.

4.4.5 HCI Commands for BISes

As with Connected Isochronous Streams, a Host application can define the SDU interval, the maximum SDU size and the Maximum Transport Latency, which is the maximum time allowed for the transmission of an SDU within a BIS. As with a CIG, it is up to the scheduler in the Controller to use these to guide it in defining the actual Link Layer parameters, i.e., NSE, BN, IRC and PTO.

Setting up a BIG is a much simpler process than setting up a CIG, largely because there is no communication with any of the receiving devices. It is a unilateral decision made by the Broadcast Transmitter, where only one HCI command is required. The LE_Create_BIG command configures and creates a BIG, with a Num_BIS number of BISes within it. There are no options to add or remove individual BISes – everything is done in the single

Section 4.4 - Broadcast Isochronous Streams

LE_Create_BIG operation. To terminate the BIG, the LE_Terminate_BIG HCI command ends and removes it. Both commands apply only to an Initiator. Multiple BIGs can be created and transmitted at the same time if there are sufficient resources and airtime. These act independently of each other.

There are two similar commands for an Acceptor which wants to receive one or more BISes from within a BIG, a process which is called Synchronizing with a BIS. They are the LE_BIG_Create_Sync Command and LE_BIG_Terminate_Sync Command. But before we get to those, we need to look at how an Acceptor can find a Broadcast Source and connect to it.

4.4.6 Finding Broadcast Audio Streams

When we looked at connected Isochronous Streams, we didn't talk about how the devices initiated the connection. The reason is that devices using Connected Isochronous Streams connect in exactly the same way as every other Bluetooth LE device, using the normal advertising and scanning procedures. They pair, bond, set up ACL links, discover each other's features and then get on with setting up the streams. If they are a Coordinated Set, CAP procedures ensure that all members of that set have the same procedures applied to them. With broadcast Audio Streams, none of that pairing and negotiation process happens, because there is no connection. Instead, receiving devices need to find a way to discover what is being transmitted, when it is being transmitted, and work out how to synchronize to it.

The method for doing this is called Extended Advertising and was added to Bluetooth® Core 5.0. Returning to basics for a moment, Figure 4.34 shows the channels which are used for Bluetooth LE. For Bluetooth LE, the 2.4 GHz spectrum is divided into 40 channels, each 2 MHz wide. Three of these are reserved for advertising at fixed frequencies - channels 37, 38 and 39 and are known as the Primary Advertising Channels.

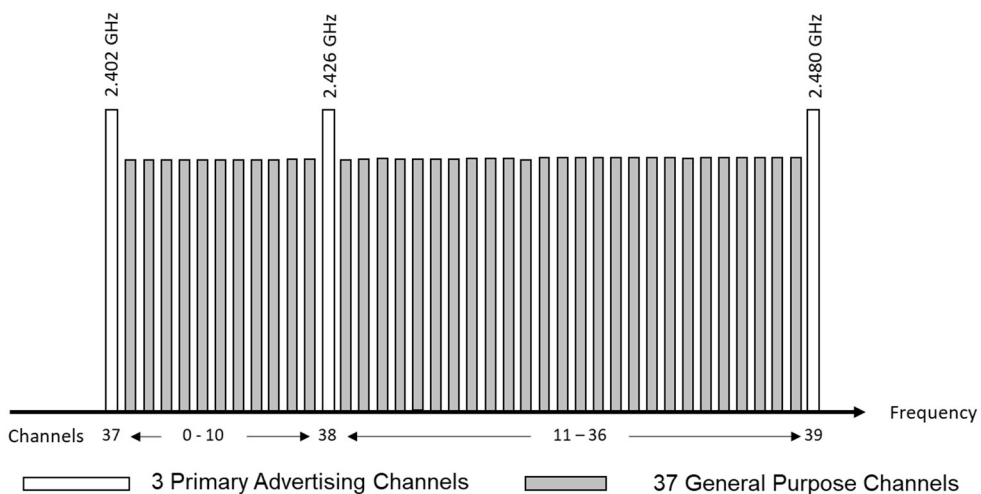


Figure 4.34 Bluetooth® LE channels

The position of these three channels was chosen to avoid the most commonly used parts of the Wi-Fi spectrum. In between the three advertising channels are 37 general purpose channels, numbered from 0 to 36, which are normally used for the transmission of data.

Broadcast Sources need to provide a fair amount of information if other devices are going to be able to receive their transmissions. Not only do they need to tell the receivers where the transmissions are located in terms of hopping channels, but they also need to provide all of the details about the structure of the BIG and its constituent BISEs, which we have just covered. In many situations, there are likely to be multiple Broadcast Transmitters operating within range of an Acceptor, particularly where they're being used in public places, and hence multiple broadcast Isochronous Streams for that Acceptor to choose from. Acceptors need to be able to differentiate between them, implying that the broadcast advertisements contain information about the content of each broadcast stream. All of this requires a large amount of information to be conveyed by the Broadcast Source. It is too much to fit into the three primary advertising channels, i.e., channels 37, 38 and 39, without the risk of overloading them.

To overcome this limitation, the Extended Advertising scheme of Bluetooth® Core 5.0 allows advertising information to be moved to the general-purpose channels - channels 0 to 36. To accomplish this, a Broadcast Source includes an Auxiliary Pointer (AuxPtr) in its primary advertisements, which informs devices that further advertising information is available in the general-purpose channels (which now serve as Secondary Advertising channels). The Auxiliary Pointer provides the information that scanning devices need to determine where these secondary advertisements are located. At this point a scanner doesn't know whether the AuxPtr is for a broadcast or something else³⁸. That only becomes apparent when it finds and reads the Extended Advertisement. Figure 4.35 illustrates the basic structure of this scheme.

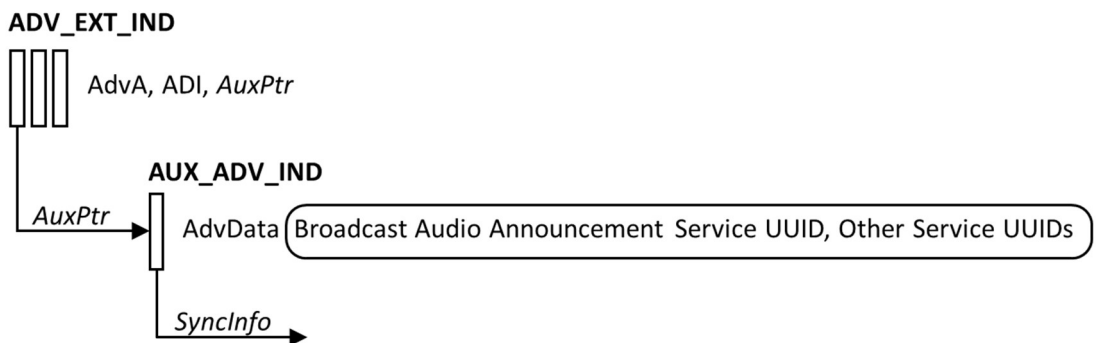


Figure 4.35 Extended advertising for a BIG

³⁸ Isochronous Channels were designed for any type of data, not just audio.

Section 4.4 - Broadcast Isochronous Streams

The Extended Advertising process is built up using two types of advertising PDU:

- **ADV_EXT_IND:** Extended Advertising Indications, which use the primary advertising channels. Each BIG requires its own advertising set and **ADV_EXT_IND**. The headers of each **ADV_EXT_IND** include the advertising address (AdvA), the ADI (which includes the Set ID for this set of Extended Advertisements), and an Auxiliary Pointer to the:
- **AUX_ADV_IND:** Auxiliary Advertising Indications. These are transmitted on the 37 general purpose channels. They contain the Advertising Data specific to a BIG, signalling that if it is an audio broadcast, along with some high level information defined by the Public Broadcast Profile (which we'll come to later).

The presence of the Auxiliary Pointer (AuxPtr) in an **ADV_EXT_IND** packet indicates that some or all of the advertisement data can be found in an Extended Advertisement. However, the amount of information that's needed to describe and locate the broadcasting stream is still bigger than what normally fits into an Extended Advertising packet. To accommodate this, the Extended Advertisement could chain further packets by including an AuxPtr to an Auxiliary **AUX_CHAIN_IND** PDU, but it doesn't. Instead, it sets up a Periodic Advertising train, which includes all of the information about a BIG and its BISes within an **AUX_SYNC_IND** PDU – an Auxiliary Synchronization Information, which is the Periodic Advertisement.

The **AUX_ADV_IND** includes a SyncInfo field, which tells scanners how to locate the Periodic Advertising. The structure of the Periodic Advertising is shown in Figure 4.36.

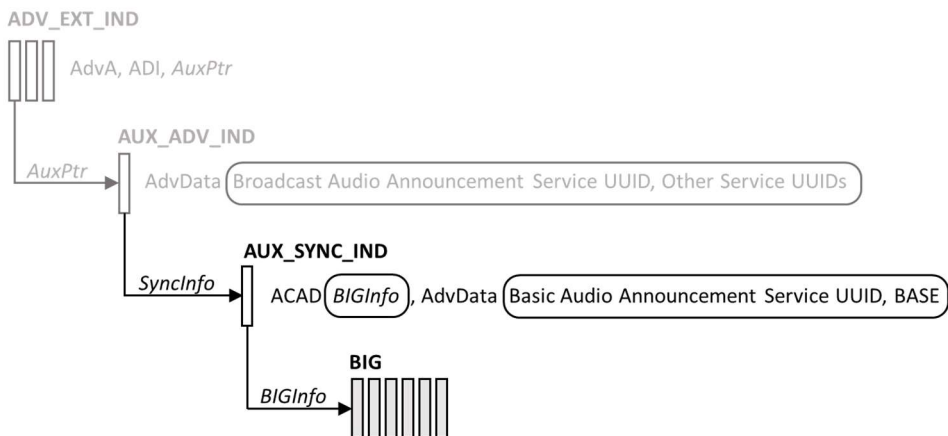


Figure 4.36 Periodic advertising for a BIG

These advertisements contain two important pieces of information. The first is the Additional Controller Advertising Data field (ACAD). This is information which is required by the Controller of the receiving device, describing the structure of the BIG and its constituent BISes. The second is information for the Host of the receiving device, which is contained in

the AdvData field. This contains the Basic Audio Announcement Service UUID and the BASE, which contains a detailed definition of the streams in the BIG, including the codec configuration and metadata which describes the use cases and content of the audio streams in a user readable format.

The AUX_SYNC_IND packets in a Periodic Advertising train are transmitted regularly, so once a scanning device has found them, it can continue tracking them, without having to refer back to the Primary or Extended Advertisements, which saves power. The Broadcast Source can turn off its Extended Advertisements, but keep the Periodic Advertising train running.

Recapping how this process works, the data needed to locate and receive a broadcast Audio Stream starts in the Extended Advertisements. The AUX_ADV_IND contains the Broadcast Audio Announcement Service UUID, which identifies the Extended Advertisements as belonging to a specific broadcast Audio Stream, with a 3 octet Broadcast_ID which identifies it. The Broadcast_ID remains static for the life of the BIG. The AUX_EXT_IND may include additional Service UUIDs from top level profiles. These allow a scanning device an early opportunity to filter the different broadcasts it finds, so that it doesn't need to synchronize with the PA and decode the information it carries for broadcasts that are not of interest to it. That feature is particularly useful to help choose public broadcasts, which are defined in the Public Broadcast Profile (PBP), which defines the Public Broadcast Announcement UUID.

Having decided it likes the look of the BIG, a scanner will synchronize to the Periodic Advertising train. These advertisements use the general-purpose channels 0 to 36. Like Extended Advertisements, they can be chained to include more data, if required, using another AuxPtr. For broadcast audio, they contain two vital pieces of information.

The first is the Additional Controller Advertising Data field, the ACAD. The ACAD contains advertising data structures. These are data structures containing an Advertising Data (AD) type with its associated data. All devices which are actively broadcasting will use the ACAD to send the BIGInfo structure, which provides all of the information needed by a receiving device to detect the broadcast and understand the BIS timings. This is the information which would be delivered to Acceptors at the Link Layer in a CIG, but with no direct connection in broadcast, that's not possible. Hence this alternative method.

Section 4.4 - Broadcast Isochronous Streams

BIGInfo				
BIG_Offset (14 bits)	BIG_Offset_Units (1 bit)	ISO_Interval (12 bits)	Num_BIS (5 bits)	NSE (5 bits)
BIGInfo (continued)				
BN (3 bits)	Sub_Interval (20 bits)	PTO (4 bits)	BIS Spacing (20 bits)	IRC (4 bits)
BIGInfo (continued)				
Max_PDU (8 bits)	RFU (8 bits)	SeedAccessAddress (32 bits)	SDU_Interval (20 bits)	Max_SDU (12 bits)
BIGInfo (continued)				
BaseCRCInit (16 bits)	ChM (37 bits)	PHY (3 bits)	bisPayloadCount (39 bits)	Framing (1 bit)
BIGInfo (continued)				
GIV (8 octets)	GSKD (16 octets)			

Figure 4.37 The BIGInfo structure

Figure 4.37 shows the structure of the BIGInfo. It starts with the BIG_Offset, which informs devices exactly where they can find the BIG in relationship to this AUX_SYNC_IND PDU. Following that, the BIGInfo describes the structure of that BIG:

- The fundamental spacing – the ISO_Interval, the Sub_Interval and the BIS Spacing
- The number of BISes in the BIG (Num_BIS) and their features - NSE, Burst Number, PTO, IRC and Framing,
- The packet information - Max_PDU, SDU_Interval, Max_SDU and bisPayloadCount.
- The physical channel access information, in the form of the SeedAccessAddress, the BaseCRCInit, the Channel Map (ChM), and the PHY, as well as the
- The Group Initialization Vector (GIV) and the Group Session Key Derivation (GSKD) to allow encrypted broadcast streams to be decoded. If these last two fields are present, a scanning device knows that the broadcast is encrypted, which means it will need to acquire the Broadcast_Code (described below) to decrypt the audio streams.

The presence of the GIV and GSKD is an easy way to tell whether a BIG contains encrypted BISes. If they are absent, the BIGInfo is shorter, indicating that the BISes are not encrypted.

Before we leave BIGInfo, there is one important thing to repeat, which is that all of the BISes have exactly the same parameters. This is in contrast to CIGs, where multiple CIGs can have different parameters. In a BIG, one setting applies to every BIS that is being used, which has to be specified to meet the maximum requirements of the different BISes. The only exception is the Control Subevent. These have a variable length, described by their BIS PDU header.

The BIG must be structured to allow for the maximum anticipated size of any Control Subevent (which could be bigger than the audio data PDUs in the BISes).

4.4.7 Synchronizing to a Broadcast Audio Stream

Having acquired the BIGInfo, devices can use this information to find out where those BISes are. The BIG_Offset, together with the BIG_Offset_Units tells receivers where the Anchor Point of the first BIS will be located after the start of the BIGInfo packet (remember that with Broadcast, there is no ACL to provide a timing instant, as there is with a CIS). Figure 4.38 shows how this information is used.

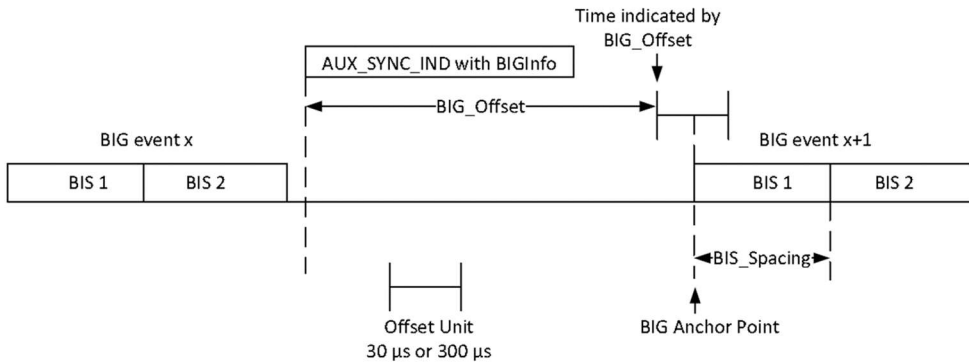


Figure 4.38 Synchronization timing for a BIG

An Acceptor can receive any number of BISes within a BIG. In Bluetooth LE Audio, this is called synchronizing with a BIG or a BIS. In most real life situations, a Broadcast Sink would not want to synchronize with all of the BISes. An earbud or hearing aid would normally only want to synchronize to the left, right or mono Audio Streams of a BIG, even if all three were present in a BIG, whereas a pair of headphones would want to synchronize to both the left and right channels.

To determine which BIS or BISes to connect to, a scanning device needs to look at a second important piece of information which is included in the Periodic Advertising AUX_SYNC_IND PDUs. This is the Broadcast Audio Source Endpoint structure (BASE), which is included in the AdvData field of each AUX_SYNC_IND PDU, following the Basic Audio Announcement Service UUID.

4.4.8 BASE – the Broadcast Audio Source Endpoint structure

To understand the BASE, we need to move up into the Basic Audio Profile, which is the first profile that we'll encounter within the Generic Audio Framework (GAF). It is the key profile in terms of configuring and managing Audio Streams, both for Connected Isochronous Streams and Broadcast Isochronous Streams.

Section 4.4 - Broadcast Isochronous Streams

BIGInfo describes the structure of the BIG, telling devices how many BISes it contains and where to find them, but that is just the practical information about the fact that they exist. It doesn't provide any information about what is in the broadcast Audio Stream, or how the different BISes are related to each other. The BASE provides that detail, telling devices what audio information is contained in each of the BISes and how it is configured. Where more than one Broadcast Source is within range, that's vital if a receiver is going to be able to choose which one it listens to. The BASE consists of three levels of information, as shown in Figure 4.39. We will revisit this in more detail when we look at how to set up a Broadcast Stream in Chapter 8.

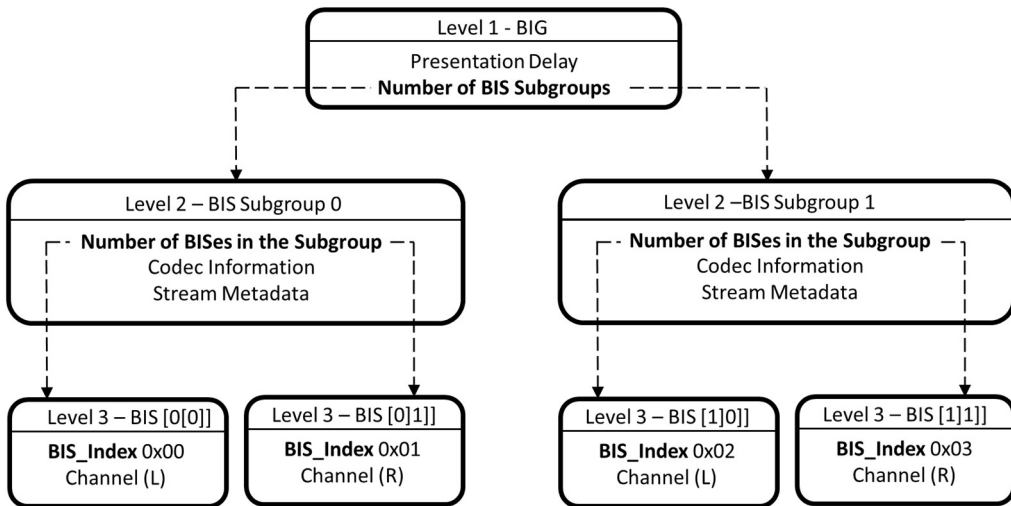


Figure 4.39 Simplified BASE structure

The highest level – Level 1, provides information which is valid for every BIS in the BIG – a single Presentation Delay and the number of Subgroups that the constituent BISes are divided into. These Subgroups contain BISes which have common features, which may be the way they are encoded, or the language of the Audio Streams.

Each Subgroup is described in more detail at Level 2 of the BASE, where the codec information is provided for the BISes in that Subgroup, as well as stream metadata in the form of LTV structures. Much of this is data strings in human readable form, such as programme information, and it's recommended that the metadata includes the Broadcast Name as a minimum, so that a scanning device which is capable of displaying it can use it to help a user select between different audio streams. This is also where a language LTV would be provided.

Finally, Level 3 of the BASE provides specific information for each BIS. This includes its BIS_Index, which identifies the order in which it appears within the BIG, as well as its Audio_Channel_Allocation LTV, which identifies the Audio Location(s) it represents, such as Left, Right or Mono. Level 3 can include a different set of codec information for specific BISes, which will override the Subgroup value of Level 2. Normally, any BIS with different

codec parameters would be placed in a separate subgroup. If it involves different sampling frequencies, it is normally more efficient to put in in a separate BIG.

The information contained in the BASE allows a Broadcast Sink to determine whether it wants to receive a BIG and which of the BISes contained within that BIG it would like to synchronize to, as well as telling it where that specific BIS is located within the overall BIG. Once it has parsed the BIGInfo and BASE, the Acceptor has all of the information it needs to synchronize to any of the BISes

In most cases, a Broadcast Source will only transmit a single BIG, but it can transmit multiple BIGs. This might occur with public information broadcasts, where different types of messages might be contained in different BIGs. For example, an airport might use one BIG for flight announcements, a second for general security announcements and a third, encrypted one for staff announcements. In this case, each BIG will have its own set of primary advertisements, each with a different Advertising Set ID (SID), along with its own extended advertising train containing its specific Broadcast_ID, BIGInfo and BASE (see Figure 4.40).

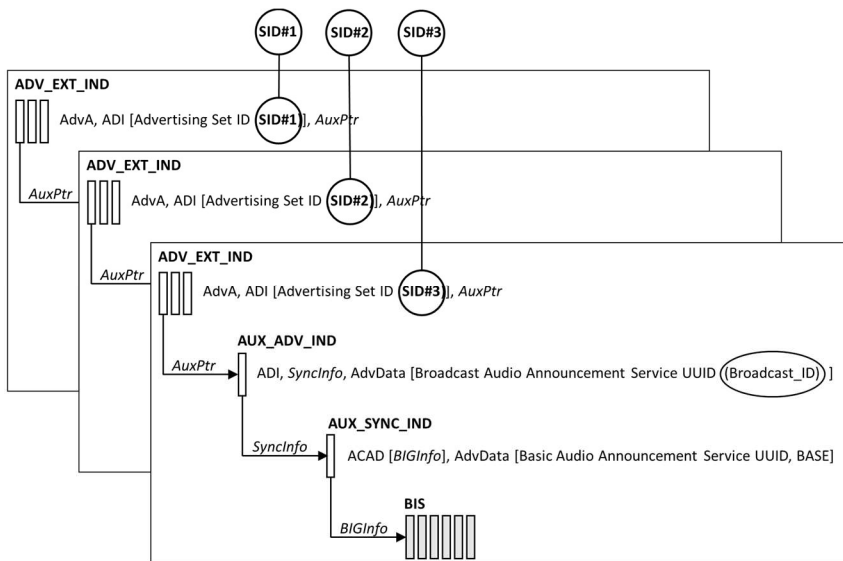


Figure 4.40 Advertising for multiple BIGs

The SID should not change often – it is typically static until a device goes through a power cycle and is often static for life. The Broadcast_ID is static for at least the life of the BIG. Broadcast Sinks can take advantage of these IDs to reconnect to a Broadcast Source that they know if they recognise the SID and Broadcast_ID.

The lack of any connection between a Broadcast Source and the Broadcast Sinks means that a transmitter has no idea of their capabilities. This can result in a Broadcast Source choosing parameters which local receivers are unable to support. Designers need to be aware of the likely capabilities of the devices a Broadcast Transmitter plans to support, particularly in terms of codec settings and the values of Presentation Delay. The Public Broadcast Profile (PBP)

Section 4.4 - Broadcast Isochronous Streams

and the Auracast™ Simple Transmitter Best Practices Guide published by the Bluetooth SIG contain recommendations to help ensure interoperability, particularly for public broadcast applications. They are a good starting point for product designers.

4.4.9 Helping to find broadcasts

The expectation is that broadcast audio will be used in a very different way to unicast audio. The hearing aid industry is keen that Bluetooth LE Audio broadcast is used to complement and extend current telecoil infrastructure, making installation more cost effective for venues. Today, most telecoil usage is for sound reinforcement, specifically for people wearing hearing aids. In the future, as the same broadcast transmissions can be picked up by consumer earbuds and headphones, far more public audio information services are likely to be deployed, as they enhance accessibility for everyone. It's also likely that Bluetooth LE Audio broadcast will become standard in TVs, both in public locations, such as gyms and bars, as well as at home. The Bluetooth SIG's Auracast™ program is promoting broadcast as a solution for shared music in cafes and for personal music.

If the market develops in this way, users will often find themselves within range of multiple Broadcast Sources, which will mean they need to choose which one to receive. A first level of selection may be provided by profile UUIDs in AUX_ADV_IND packets, but devices will still need to detect and parse the BASE information for each BIG to make a decision. (In the early days, it may be possible to select the first BIG you find, then press a button to move to the next, but that is not a scalable user experience for the long term.) This presents product designers with a problem, as devices like earbuds have no room for a display and often barely room for any buttons. In addition, the process of scanning is relatively power hungry – constant scanning would have a noticeable impact on an earbud or hearing aid's battery life.

To get around this limitation, the Bluetooth LE Audio specifications have introduced the concept of a Commander – a role which performs the scanning operation on behalf of its Acceptor. It allows a user to select a broadcast and then instruct a receiving device to synchronize to the selected BIS or BISes. The Commander role can be implemented as part of an app in a mobile phone, in a smart watch, an earbud case or a dedicated remote-control device. In fact, any Bluetooth LE device which has a connection with a Broadcast Sink can act as a Commander. Devices which implement the Commander role are called Broadcast Assistants and are defined in the Broadcast Audio Scanning Service (BASS). They can be integrated into devices like TVs to help automate the connection to earbuds and headphones.

4.4.10 Periodic Advertising Synchronization Transfer – PAST³⁹

A Broadcast Assistant scans for advertisements which indicate the presence of Extended

³⁹ The Core does not use the acronym PAST for Periodic Advertising Synchronization Transfer, which was introduced in BAP. So, if you're searching the Core, you need to look for "Periodic

Advertisements in exactly the same way as any other scanning device. Once it discovers them, it can synchronize to the associated Periodic Advertising train, which contains a Broadcast Audio Announcement Service UUID and then discover the accompanying BIGInfo and BASE structure. It may apply filters to its scan before reading and parsing the BASE metadata of those it finds, after which it provides the list of available broadcast streams to the user, generally by displaying the human readable Broadcast Name.

Once the user has made their selection, the Broadcast Assistant will use the PAST procedure to provide the Broadcast Receiver with the information it needs to find the relevant Periodic Advertising train. This allows the Broadcast Receiver to jump straight to those advertising packets, acquire the BIGInfo and BASE and synchronize to the appropriate BISes without having to expend energy in scanning. This process is called Periodic Advertising Synchronization Transfer or PAST and is defined in the Core, Vol 6, Part B, 5.1.13.

The level of information provided by a Broadcast Assistant to the user is entirely down to the implementation. A phone app could list every Broadcast Source within range; it could limit the amount of information displayed based on user preferences, or use preconfigured settings which it had read from a set of earbuds. The Broadcast Assistant could equally be a button on a watch or fitness band which selects the last known synchronized stream from the list of Broadcast Sources it has found. Broadcast Assistants can also include applications to obtain the required Broadcast_Code to decrypt private, encrypted broadcasts, either by making a Bluetooth connection to the Broadcast Source or a proxy, or by using an out of band (OOB) method, such as scanning a QR code.

4.4.11 Broadcast_Code

Encrypted streams are not new in Bluetooth technology; security and confidentiality have always been an important part of the specifications. However, implementing encryption when there is no connection between the transmitting and receiving devices, as is the case with the broadcast streams in Bluetooth LE Audio, introduces a new problem, which is how to acquire the encryption key.

Many broadcast streams will not be encrypted; they will be broadcast openly, so that anyone can pick them up. However, that generates a few issues. Firstly, anyone who is within range can receive them. As Bluetooth technology is quite efficient in penetrating walls, that can be an issue in meeting rooms, hotel rooms and even homes, where someone in a neighbouring room could inadvertently pick up the audio from your TV. That could range from annoying to embarrassing, depending on what the content is. In a business environment, it may well be

Advertising Sync². Note that this will bring up the Periodic Advertising Synchronization Transfer procedure [Vol 3, Part C, 9.5.4], which is the GAP procedure referred to in BAP and BASS, as well as the Periodic Advertising Sync Transfer procedure [Vol 6, Part B, 5.1.13], which is the underlying Link Layer procedure).

Section 4.4 - Broadcast Isochronous Streams

confidential, so adding encryption is vital. Basic confidentiality is inherent within telecoil systems, as the audio transmission can only be picked up within the confines of the induction coil. To provide the same level of authentication in Bluetooth LE Audio broadcasts, the audio data needs to be encrypted, which requires the receiver to obtain the decryption key, which is known as the Broadcast_Code.

Devices looking for broadcasts can detect whether a broadcast Audio Stream is encrypted by examining the length of the BIGInfo in the periodic advertising chain. If the stream is encrypted, the packet will include an additional 24 octets, containing a Group Initialisation Vector (GIV) and a Group Session Key Diversifier (GSKD). Scanners will detect their presence and, in conjunction with other metadata, determine whether or not to synchronize with such a stream, depending on whether they are able to retrieve the Broadcast_Code.

Although broadcast does not need a Broadcast Source and Sink to be paired, in many cases there will be an ACL connection present. That may sound strange, but it's an arrangement that allows the number of encrypted connections to be scaled far beyond what is possible with unicast, without hitting an airtime limit. This is expected to be the way most domestic TVs will work, so that neighbours can't hear what you are listening to, but multiple authorised listeners can hear the audio at the same time. In this case the users would be paired to the TV, using the features of BASS to obtain the Broadcast_Code. The Broadcast_Code is a property of the Host application. The Host provides it to the Controller when it is setting up the BISes, and it can equally supply it to trusted devices. In public spaces, conference rooms and hotels, it is likely that an out-of-band method would be used, probably similar to the printed information which is used to connect to a Wi-Fi access point. Broadcast_Codes can also be obtained through other out of band methods, such as scanning QR codes, tapping an NFC terminal, or even being included with a downloadable theatre ticket. We'll explore these configurations in more detail in Chapters 12 and 14

Note that the metadata used to describe broadcasts within advertising packets is not encrypted. Accordingly, care should be taken to ensure that it does not contain confidential or potentially embarrassing information.

4.4.12 Broadcast topologies

The features included in Bluetooth LE Audio provide a wide range of options for finding broadcasts. We will look at them in more detail in later chapters, but the figures below show some of the common ones. In most cases, they show a single "Broadcast Information" stream, which encompasses all of the advertising information

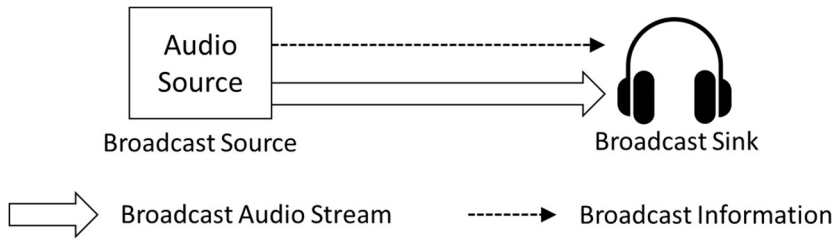


Figure 4.41 Direct synchronization from headphones

The simplest case, where a pair of headphones does its own scanning to find and select a broadcast audio stream, is shown in Figure 4.41.

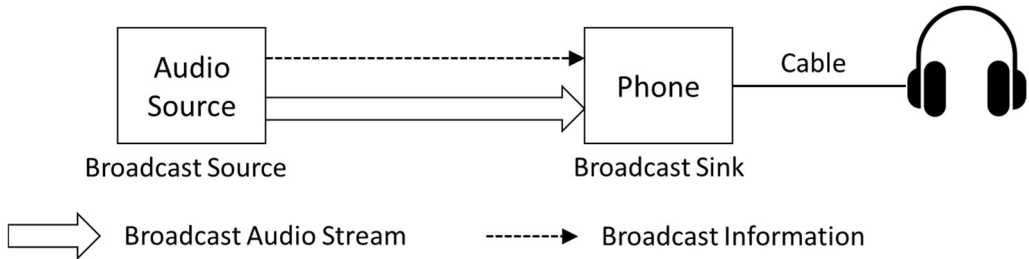


Figure 4.42 Using a phone as a Broadcast Sink

Figure 4.42, is essentially the same, but points out the Broadcast Sink could also be a phone. Here, it can scan for Broadcast Sources, display the available choice of broadcast Audio Streams to the user and then render the audio to a pair of wired headphones. It could equally transmit the streams using Bluetooth (either Bluetooth Classic Audio or Bluetooth LE Audio), although that is unlikely to be an efficient use of airtime.

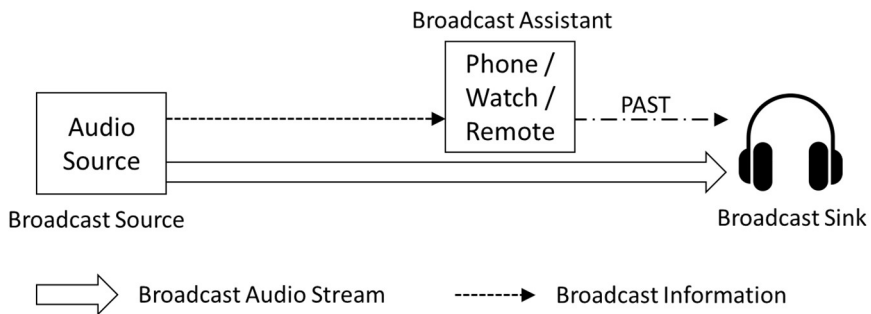


Figure 4.43 Using a Broadcast Assistant and PAST

Figure 4.43 shows what is expected to be the most common use case, where a device like a phone, watch or remote control acts as a Broadcast Assistant to scan and present the choice of broadcast Audio Streams to the user, then uses PAST, the Periodic Advertising Synchronization Transfer feature, to allow the user's headphone or earbuds to connect to the selected stream. As we will see, the Broadcast Assistant can also be used for volume control and mute, allowing a user to find, select and control the rendering of a broadcast Audio Stream. It is illustrated in Figure 4.44 which shows how an app on a mobile phone could be used both

Section 4.4 - Broadcast Isochronous Streams

to display and select broadcast streams, as well as controlling volume and balance on a pair of earbuds.

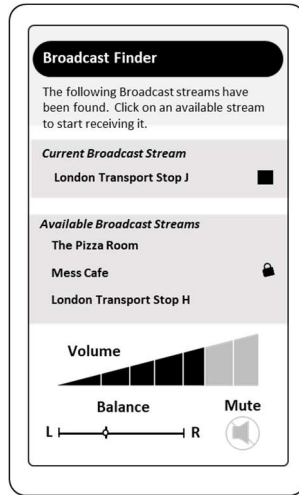


Figure 4.44 Example of a phone app being used as a Broadcast Assistant and Volume Controller

Although there is no connection between a Broadcast Source and a Broadcast Sink for the Audio Stream, devices can use ACL connections to help synchronize to the BIS. An example of this is a domestic TV which uses encrypted, broadcast audio to allow multiple people in the room to connect to it. To achieve this, the TV would include a Broadcast Assistant, which would be paired to each user's headset, as shown in Figure 4.45. When a user chooses to connect to the TV, the Broadcast Assistant would provide details of how to synchronize to the BISes using PAST, as well as transferring the Broadcast_Code. This could be achieved by a button press on a headset, or just by proximity, as the user comes within range of the TV's Broadcast Assistant.

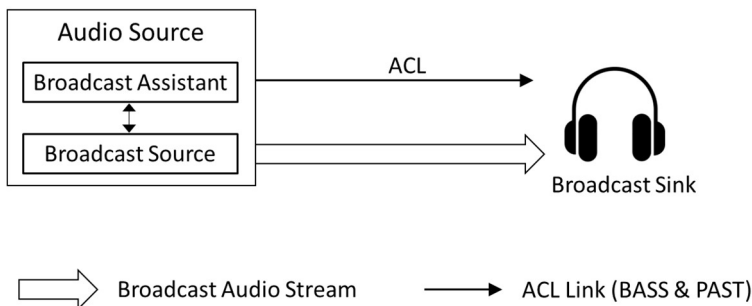


Figure 4.45 Collocation of a Broadcast Assistant with a Broadcast Source

This is one of the topologies that forms the basis of audio sharing, where a number of friends can share music from one phone. Chapter 12 explains these options in greater detail.

4.5 ISOAL – The Isochronous Adaptation Layer

ISOAL [Core Vol 6, Part G] is one of the most complicated aspects of the Core Isochronous Streams feature. The good news is that it's taken care of for you in the Bluetooth chips, but a knowledge of why it's necessary and what it does is useful. ISOAL is used for both broadcast and unicast Audio Streams.

ISOAL exists to solve the problem of what happens when there is a mismatch between the transmissions intervals used for different devices connected to an Initiator. The most common reason for this is that an Acceptor only supports a 10ms frame size (for which it would like to use 10ms Isochronous Intervals), but an Initiator needs to use 7.5ms intervals, as it is running connections with other Bluetooth devices, such as older mice and keyboards, which are only capable of running at a 7.5ms timing interval⁴⁰. That will change in time, as new peripheral devices become more flexible, but until then, ISOAL provides a means to accommodate them while the whole Bluetooth ecosystem migrates to a 10ms timing interval.

The ISOAL layer sits in the data path between the codec and the Link Layer. Encoded SDUs from the codec may be delivered via the HCI if the codec is implemented in the Host, or through a proprietary interface (regardless of where the codec is situated). As Figure 4.46 shows, ISOAL provides fragmentation and recombination or segmentation and reassembly and is responsible for sending the encoded audio data in either framed or unframed PDUs.

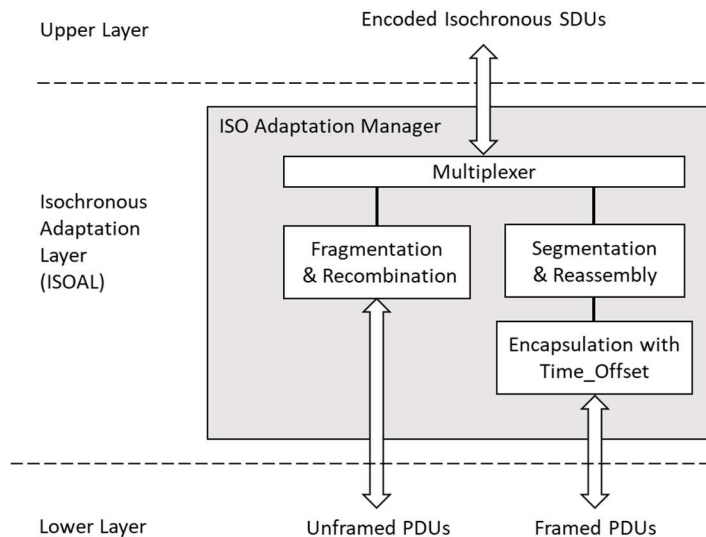


Figure 4.46 Architecture of the Isochronous Adaptation Layer (ISOAL)

⁴⁰ Classic Bluetooth is based on a 2.5ms interval, but many applications standardised on 7.5ms, which causes this problem for dual-mode devices.

Section 4.5 - ISOAL – The Isochronous Adaptation Layer

Depending on the setting provided by the Host layer, the Controller can decide whether to use framed or unframed PDUs. For unframed PDUs, if an SDU can fit within a single PDU, the Isochronous Adaptation Manager may fragment them, but sends them without a segmentation header. That's the most efficient, lowest latency way to transport Isochronous data. If the SDU is larger than the Maximum PDU size, it will be segmented and sent in multiple unframed PDUs. The recombination or reassembly processes reassemble them back into SDUs. Unframed PDUs can only be used when the ISO_Interval is equal to or is an integer multiple of the SDU_Interval, which is itself equal to or an integer multiple of the sampling frame. This means that the generation of the SDUs needs to be synchronized with the transport timing, so that they don't drift with respect to each other. Otherwise, you need to use framed SDUs.

For framed SDUs, the Isochronous Adaptation Manager adds a segmentation header and an optional Time_Offset. The Time_Offset allows multiple SDUs to be segmented across PDUs, providing a reference time that maintains an association between the SDU generation and transport timing. If you need more detail, the full specification of ISOAL is contained in Vol 6, Part G, Section 6 of the Core.

The main aspect of ISOAL that designers need to be aware of is that this section of the Core specification contains the set of equations which define the Transport_Delay. Transport_Delay is a key component of the overall latency, being the time between an SDU being presented for transmission and the point where it is ready for decoding at the appropriate Synchronization Reference.

For framed SDUs, the CIG transport latencies are:

$$\text{Transport_Latency} = \text{CIG_Sync_Delay} + \text{FT} \times \text{ISO_Interval} + \text{SDU_Interval}$$

Separate calculations need to be made using the respective values of CIG_Sync_Delay, FT and SDU_Interval for the Central to Peripheral and Peripheral to Central directions.

For a BIG using framed SDUs,

$$\text{Transport_Latency} = \text{BIG_Sync_Delay} + (\text{PTO} \times (\text{NSE} \div \text{BN} - \text{IRC})) \times \text{ISO_Interval} + \text{ISO_Interval} + \text{SDU_Interval}$$

For unframed SDUs, the calculations are slightly different. For a CIG:

$$\text{Transport_Latency} = \text{CIG_Sync_Delay} + \text{FT} \times \text{ISO_Interval} - \text{SDU_Interval}$$

Again, you need to use the respective values of CIG_Sync_Delay, FT and SDU_Interval for the Central to Peripheral and Peripheral to Central directions.

For an unframed BIG,

$$\text{Transport_Latency} = \text{BIG_Sync_Delay} + (\text{PTO} \times (\text{NSE} \div \text{BN-IRC}) + 1) \times \text{ISO_Interval} - \text{SDU_Interval}$$

--oOo--

That concludes the basics of Isochronous Streams. Now we need to look at the LC3 and see how Quality of Service (QoS) choices influence robustness and latency, as that is an important influence on how BISes and CISes are configured.

Chapter 5. LC3, latency and QoS

5.1 Introduction

Two of the most debated aspects of wireless audio, particularly amongst audiophiles, are audio quality and latency. In its early years, Bluetooth® technology was often criticised for both, although in most cases the audio quality probably had more to do with the state of the transducers rendering the audio than anything to do with the Bluetooth implementation or specifications.

Transmitting audio over any wireless connection involves compromises. Interference is a fact of life, which means that some of the audio data will be lost. That results in gaps in the audio unless you take measures to add redundancy. Typically, that involves transmitting the audio packets more than once, so that there are multiple chances that one of them will get through. However, to be able to do that, you have to be able to compress the audio, so that you have time to transmit multiple copies. That's done using codecs (which is a portmanteau word for coder and decoder).

The coder takes in an analogue signal, digitises it and compresses the digital data, so that it can be transmitted in a shorter time than the length of the original sample. This means that it can be transmitted multiple times before the next sample is taken. If the first transmission is lost or corrupted, the following retransmission can be used in its place. The decoder, in the receiving device, decodes the received data, expanding it to regenerate the original audio signal. As it takes time to perform the encoding and decoding, this results in a delay between the original signal and the reconstituted signal coming out of the decoder.

Audio codecs are a relatively recent invention. The first hundred years of audio transmission, from the 1860's phonoautograph, through radio broadcasts, vinyl records and magnetic tape, all worked with the original audio signal. If there was interference from the weather, a scratch on a record or wax cylinder, or stretching on a tape, the sound was lost or distorted. This changed with the introduction of CDs, which were made possible by the development of pulse code modulation (PCM), which converts an analogue signal to a digital signal.

PCM works by sampling the audio signal at a higher frequency than we can hear (44,100 times per second for a CD), converting each sample into a digital value. Decoding performs this operation in reverse, using a digital to audio decoder to restore the analogue signal. The more bits in each sample, the closer the output audio will be to the original input. CDs, along with most audio codecs, use 16 bit samples. Where the sampling rate and the bits per sample are sufficiently high, the human ear can't detect the difference. However, the file sizes for a pure PCM digital file are large, as there is no compression involved. Sampling at 44.1kHz and 16bits generates 800kbits every second, so a five minute song in mono is around 26MB, or 52MB for stereo. That's what limits a standard CD to holding around an hour of music.

The arrival of the MP3 audio codec, developed by the Fraunhofer institute, transformed the distribution of digital music. It uses a technique called perceptual coding (sometimes also called psychoacoustic modelling) which compares the audio stream with a knowledge of what a human ear can actually hear. That may be a high frequency sound which is above the range most people can hear, so can be encoded with less data, or a held note, where an encoder can indicate that you just need to repeat the previous sample or encode a difference from it. By applying these methods, it is possible to significantly reduce the size of a digitised audio file. An MP3 codec typically reduces the size of a digitised music file by between 25% and 95%, depending on the content. Few listeners were worried about any slight loss of quality from this process, feeling that the increased convenience far outweighed any noticeable effect on the music.

The reduction in the size of music files led to the creation of music sharing services like Napster and the appearance of MP3 players. It also fired the starting pistol for the development of streaming services and wireless audio transmission, as the reduced file sizes meant that there was plenty of time to retransmit the compressed audio packets, helping to cope with any interruptions to transmission.

5.2 Codecs and latency

One downside to the use of codecs is that they add latency to the signal. This is a delay between the arrival of the original analogue signal at a transmitter and the rendering of the reconstituted signal at the receiver.

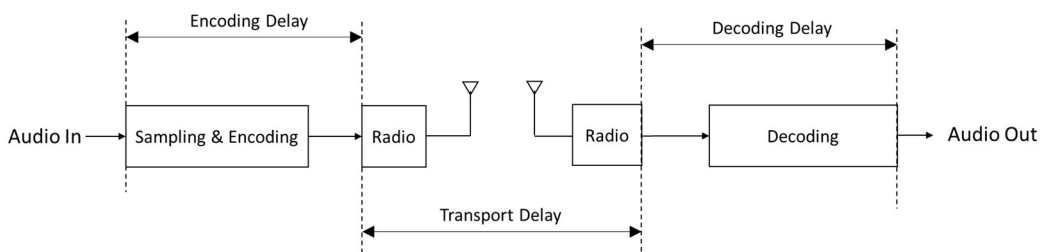


Figure 5.1 The elements of latency in an audio transmission

Figure 5.1 shows the elements that make up that latency. First, the audio is sampled. Perceptual coding requires a codec to look at multiple, consecutive samples, as a lot of the opportunities for compression come from identifying periods of repeated sound (or lack of sound). This means that most codecs need to capture sufficient, successive samples to have enough data to characterise these changes. This period of sampling is called a frame. Different encoding techniques use different frame lengths, but it's almost always a fixed duration. If it's too short, the limited number of samples starts to reduce the efficiency of the codec, as it doesn't have enough information to apply the perceptual coding techniques, which impacts the quality. On the other hand, if the frame sizes grow, the quality improves, but the latency increases, as the codec has to wait longer to collect each frame of audio data.

Section 5.2 - Codecs and latency

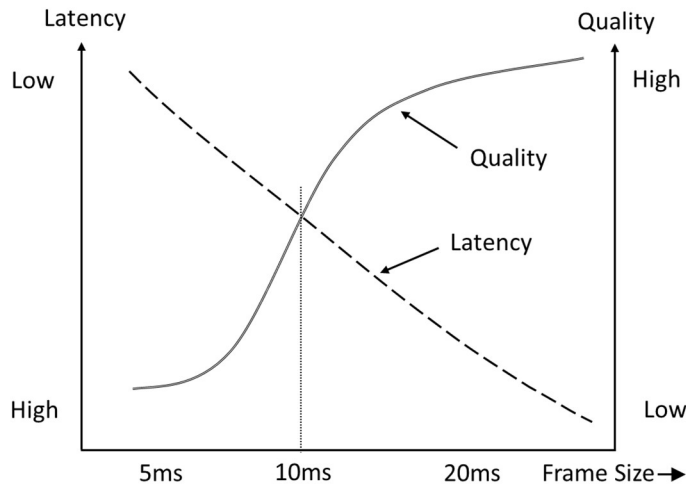


Figure 5.2 The sweet spot for audio codec frame size

Figure 5.2 illustrates the trade-off. This will vary from codec to codec, depending on how they perform their compression. But for a codec which can be used for both voice and music, the general rule is that smaller frame sizes give lower latency (which is generally considered to be better, particularly for voice and live music), but bigger frame sizes result in higher quality, as they contain more information to feed into the psychoacoustic algorithms. The industry has found that there is a sweet spot for the frame length of around 10ms, which gives good quality for voice and music at a reasonable latency.

There is another trade-off, which is the amount of processing power that you need to run the codec, which is known as the complexity. As you try to squeeze more audio quality out of the codec, you generally need a faster processor, which starts to reduce the battery life. That may not be a problem on a phone or PC, but if you are encoding the microphone input of a hearing aid or earbud, it's a very serious problem.

Returning to Figure 5.1 and the general principles of wireless audio transmission, once the audio frame has been encoded, the radio will transmit it to the receiving device. The transmission is normally quick compared to the encoding, but if the protocol contains retransmission opportunities, you need to allow for these before you start decoding. The duration between the start of transmitting the first time, to the end of the last transmission being received is called the Transport Delay and can range from a few milliseconds to several tens of milliseconds. (You can start decoding as soon as you receive the first packet, but if you do you will need to buffer it. That's because the output audio stream needs to be reconstructed to have no gaps, so it must be delayed until every opportunity for a retransmission has passed, to cope with the instances when a packet needs the maximum number of retransmissions to get through. Otherwise, packets which arrive early will be rendered early, while others won't.)

Finally, after the encoded audio data has been received, it needs to be decoded to convert it back to analogue form to be rendered. Decoding is normally quicker than encoding and doesn't have a frame delay, as the decoder expands the output frame automatically. It generally uses far less power than encoding, as most codecs are designed for use cases where a file is encoded once at production, then decoded many times (as when you're streaming music from a central server), so there is an inherent asymmetry in most audio codec designs.

5.3 Classic Bluetooth codecs – their strengths and limitations

The existing Bluetooth audio profiles were both developed with specific requirements for their individual use cases, with different codecs optimised for each, as shown in Figure 5.3. The original HFP specification was designed to use a CVSD (Continuous Variable Slope Delta modulation) coding method, which is a low latency codec, widely used in telephony applications.

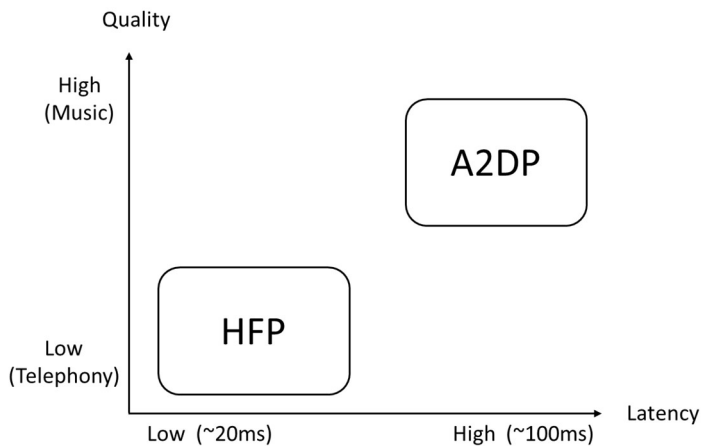


Figure 5.3 Performance of HFP and A2DP profiles

CVSD was one of the first methods for digitising and compressing voice. It samples rapidly - typically at 64,000 samples per second, but only captures the difference between the current sample and the preceding one. This means that it is frameless and has a comparatively short sampling and encoding delay. Similarly, the output decode can be performed quickly. The trade-off is that the quality is limited and because there is no compression it is effectively real-time, with no opportunity for retransmission.

Later versions of HFP include mSBC - a modified version of the SBC codec specified in A2DP, to support wideband speech. mSBC is effectively a cut down version of SBC, with a limited sampling frequency for a single, monaural stream. Being a frame-based codec, it increases the latency, resulting in typical overall delays of around 30ms. These put HFP in the low latency, low to medium quality quadrant of Figure 5.3.

Section 5.3 - Classic Bluetooth codecs – their strengths and limitations

In contrast, A2DP was designed for high quality music. It mandates the SBC (Sub Band Coding) Codec, which is a frame-based codec with fairly basic psychoacoustic modelling. It can produce very good audio quality, which is close to the limit of what an experienced listener can detect, compared to the original audio stream. The A2DP specification also allows the use of alternative⁴¹ codecs which were developed by external companies or standards groups – a selection which includes AAC⁴² (which is used by Apple in most of their Bluetooth products), MP3 and ATRAC⁴³, as well as an option for companies to use proprietary codecs. A number of these have become popular, of which the best known is the AptX range from Qualcomm. Almost all of these codecs have longer latencies.

In Figure 5.3, A2DP is in the top right quadrant of the diagram, with a long latency. That is partly driven by the desire for using retransmissions to try to make the audio more robust. Whereas listeners used to accept glitches like scratches on records, they appear to be far less tolerant of the occasional “pop” or dropout in an audio stream. The simplest solution is to add more retransmissions and buffering, but that means that wireless music streaming typically has a latency of 100 – 200 ms, even if you’re streaming from a file on your phone or computer. Although the codec isn’t involved in this delay, the knowledge that it happens means that codec designers haven’t generally concentrated on improving latency, unless it’s for a specific application like gaming.

Although a 100 – 200ms delay sounds excessive, for most music applications it’s not a problem. When streaming music, whether from a music player or an internet service, the user has nothing to indicate whether they’re listening in real time or not. As long as the music stream starts within a second of them pressing the Play button, and the music stream is continuous, without annoying interruptions, they’re happy. However, when the audio is a soundtrack for a video, they may notice a lip-synch problem, as a 200ms delay between seeing someone talking and hearing their voice looks wrong. Phone and TV manufacturers can address this by delaying the video to compensate for any audio delay. The Audio/Video Distribution Transport Protocol (AVDTP), which underlies the A2DP profile, contains a Delay Reporting feature which allows audio source devices to ask the receivers what the latency will be in the audio path. Knowing this, TVs and phones can delay the video, so that both sound and picture are synchronized. However, many TVs and earbuds have limited memory for audio or video buffering, so latencies above a few hundred milliseconds may be problematic.

Even short audio delays can become a problem where a user can hear both the Bluetooth audio and also the original, ambient source of the sound. This has long been recognised by the hearing aid industry, where users are listening to live sound via a telecoil system in a theatre

⁴¹ Referred to as “optional” in older specifications and “additional” in more recent ones.

⁴² AAC is the Advanced Audio Codec

⁴³ ATRAC is the Adaptive Transform Acoustic Coding codec, developed by Sony

or cinema, but can also hear the ambient sound. The same problem occurs at home where a family watching the TV includes some members who are wearing hearing aids with support for wireless transmission and some who are not. Telecoil induction loops, which are currently used for these applications, are analogue, so exhibit virtually no delay. To address these limitations, the Bluetooth SIG needed to find a codec that was capable of covering much more of the Quality / Latency spectrum than SBC.

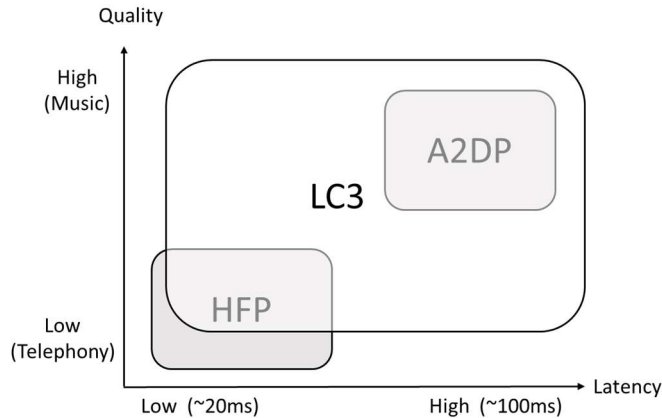


Figure 5.4 LC3 - a more efficient codec

The current SBC codec presented more limitations than just limited audio quality and latency. Although it is relatively efficient SBC with a low complexity, it takes up too much airtime, which can have a major effect on the battery life of an earbud. That's a problem for hearing aids which run on small zinc-air batteries. These are sensitive to both peak current and the length of a current burst for reception or transmission (Bluetooth chips can often consume more current receiving than transmitting.) If operating limits are exceeded for these batteries, their life can be drastically reduced. To address these limitations, the Bluetooth SIG went on a codec hunt, which resulted in the inclusion of the new Low Complexity Communications Codec, more commonly known as LC3.

Bluetooth LE Audio allows manufacturers to use other codecs, but LC3 is mandatory for all devices. The reason for this is to ensure interoperability, as every Audio Source and every Audio Sink has to support it. The full specification for the codec is published and falls under the Bluetooth RANDZ⁴⁴ license, so anybody can write their own implementation and incorporate it into their Bluetooth product, as long as those products pass the Bluetooth Qualification process. Given its quality, that's a powerful incentive to use it.

⁴⁴ A RANDZ licence is a Reasonable and Non-Discriminatory Zero fee license, which is how the Bluetooth IP is licensed. That doesn't mean there are no conditions, but they are not arduous. They're explained at the Bluetooth website.

5.4 The LC3 codec

The LC3 is one of the most advanced audio codecs available today, providing enormous flexibility and covering everything from voice to high quality audio. Anyone can develop their own implementation of LC3 for a Bluetooth LE Audio product, although, given the specialised nature of writing and optimising a codec, very few people are ever likely to do that. Instead they will use an implementation which comes from their silicon or stack provider. Because of that, I'll only provide an overview of what it does and how it works. There's a more detailed introduction in the LC3 specification, along with around two hundred pages of specification detail for anyone who fancies doing their own implementation⁴⁵.

The LC3 specification is among the most successful attempts so far to cover the full range of audio quality and latency requirements for wireless audio in a single codec. It is optimised for a frame size of 10ms, and it is expected that all new applications, in particular public broadcast applications, will use the mandatory 10ms frames. It also works with a 7.5ms frame size to provide compatibility with Bluetooth Classic Audio applications which run with 7.5ms intervals (corresponding to EV-3 SCO packets). In addition, it supports an extended 10.88ms frame, to provide legacy 44.1kHz sampling. This is reduced to 8.163ms for a 7.5ms based system which needs to support 44.1kHz. However, these are specific variants to support legacy, or combined Bluetooth Classic Audio / Bluetooth LE Audio implementations.

Feature	Supported Range
Frame Duration	10ms (10.88ms @ 44.1kHz sampling) 7.5ms (8.163ms @ 44.1kHz sampling)
Supported Sampling Rate	8kHz, 16kHz, 24kHz, 32kHz, 44.1kHz and 48kHz
Supported bitrates	20 – 400 bytes per frame for each audio channel. The bitrate used is specified or recommended by the Bluetooth LE Audio profiles.
Supported bits per audio sample	16, 24 and 32. (The algorithm allows most intermediate values, but these are the recommended ones.)
Number of audio channels	Single channel encoding, but multiple channels can be multiplexed within BAP. In practice, the number of audio channels is limited by the profile, implementation resources and airtime.

Table 5.1 LC3 features

Table 5.1 reproduces the key parameters from Table 3-1 of the LC3 specification. If more than one audio channel is being transmitted, one LC3 instance is normally required for each

⁴⁵ For those looking for a less complex explanation, there is an introductory video at www.bit.ly/LC3video.

audio channel and each sampling frequency. For example, a Broadcast Source transmitting stereo at 48kHz sampling would require two instances. If it included a down-mixed mono channel, it would need three instances, and if it wanted to transmit the stereo stream at both 24kHz and 48kHz sampling rates, it would require four instances.

The Bluetooth SIG has commissioned extensive audio quality testing from independent test labs to quantify the subjective performance of the LC3 codec. These show that at all sample rates, the audio quality exceeds that of SBC at the same sample rate, and provides equivalent or better audio quality at half the bitrate. The practical benefit is that the total size of LC3 encoded packets is around half of the size of those for SBC for the same audio stream. Implementers can use that to their advantage, as it reduces the total airtime for transmission, saving battery life, or they can use it to increase the audio quality. It gives them more scope to play with parameters, particularly in power constrained devices like earbuds and hearing aids. The power saving also allows scope for adding extra functionality into earbuds, such as more advanced audio algorithms or physiological sensors, whilst retaining a long battery life.

5.4.1 The LC3 encoder

Figure 5.5 provides a high-level view of the LC3 encoder.

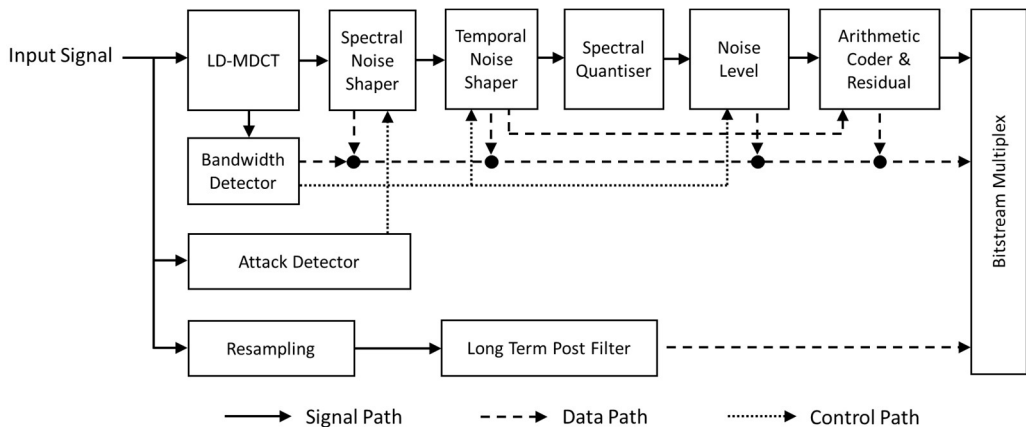


Figure 5.5 High level overview of the LC3 encoder

The first element of the encoder is the Low Delay Modified Discrete Cosine Transform module (LD-MDCT). LD-MDCT is a well-established method of performing time to frequency transformations in perceptual audio coding. It is low delay (hence the LD), but it still takes time to convert the audio input sample into spectral coefficients and group the corresponding energy values into bands. It's where a fair proportion of the codec delay comes from.

One of the modules fed from the LD-MDCT is the Bandwidth Detector, which detects incoming audio signals which have previously been sampled at different coding rates. It can detect the commonly used speech bandwidths in voice communication, i.e., NB (Narrow Band: 0-4 kHz), WB (Wide Band: 0-8 kHz), SSWB (Semi Super Wide Band: 0-12 kHz), SWB

Section 5.4 - The LC3 codec

(Super Wide Band: 0-16 kHz) and FB (Full Band: 0-20 kHz). If it detects a mismatch, it signals to the Temporal Noise Shaper (TNS) and Noise Level modules to forestall and avoid any smearing of noise into any upper spectrum.

The main path for the frequency components generated by the LD-MDCT is into the Spectral Noise Shaper (SNS) where they are quantised and processed. The job of the SNS is to maximise the perceptual audio quality by shaping the quantisation noise so that the eventual, decoded output is perceived by the human ear as being as close as possible to the original signal.

The remaining modules in the encoder are largely responsible for controlling artefacts. The most difficult sounds for a codec to handle are ones with a sharp attack, such as percussion instruments. Those transients are such a difficult thing for codecs to deal with that castanets, glockenspiel and triangles are key test sounds which are used for assessing a codec's performance. Part of the problem with sharp attack transients is that overall, these sounds show a fairly flat spectrum. The Attack Detector signals their presence to the Spectral Noise Shaper, so that it can inform the Temporal Noise Shaping module (TNS) of their presence. The TNS then reduces and potentially eliminates the artefacts for signals which have severe transients.

The next stage is to determine the number of bits required to encode the quantised spectrum, which is the job of the Spectral Quantiser. It can be considered as an intelligent form of automatic gain control. It also works out which coefficients can be quantised to zero, which the decoder can interpret as silence. This process risks introducing some coding artefacts, which are addressed by the Noise Level module, using a pseudo random noise generator to fill any gaps, ensuring that everything is set to the proper level for the decoder. It also uses the input from the bandwidth detector to ensure that the encoded signal is restricted to the active signal region. Once that is done, the spectral coefficients are entropy encoded and multiplexed into the bitstream.

One other component of the resulting bitstream is a resampled input. Performed at a fixed rate of 12.8 kHz, this is passed through a Long Term Post Filter (LTPF). For low bit rates this reduces coding noise in any frames which contain pitched or tonal information. The Long Term Postfilter (LTPF) module perceptually shapes quantization noise by controlling a pitch-based postfilter on the decoder side.

An encoded LC3 frame does not contain any timing information, such as time stamps or sequence numbers. It is up to the system using the LC3 to control the timing of packets.

5.4.2 The LC3 decoder

The decoder is shown in Figure 5.6 and essentially reverses the process.

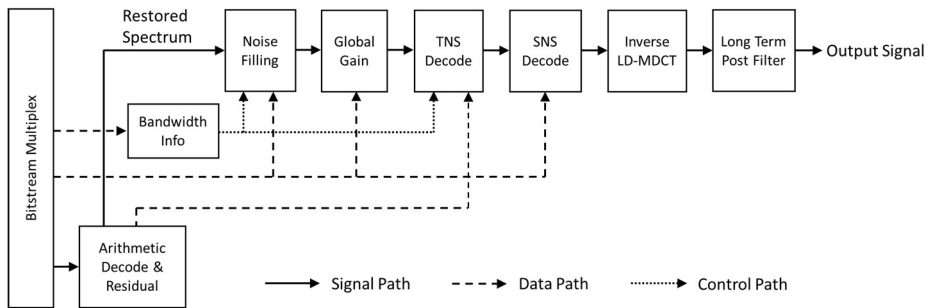


Figure 5.6 Overview of the LC3 decoder

The bandwidth information is used to determine which coefficients are zero, with the Noise Filling model inserting information for those which are inband. The Temporal Noise Shaper and Spectral Shaper process these, before the Inverse LD-MDCT module transforms them back to the time domain. The Long Term Post Filter is then applied, using the transmitted pitch information to define the filter characteristic.

Before the received packets for each frame are decoded, the Controller generates a Bad Frame Indication flag (BFI) if it detects any errors in the payload, along with a payload size parameter for each channel. If the BFI flag is set, the decoder will skip the packet and signal that a Packet Loss Concealment (PLC) algorithm should be run to replace missing data in the output audio stream. Any errors detected during the decode, will also trigger the PLC.

5.4.3 Choosing LC3 parameters

From a Bluetooth LE Audio design viewpoint, the closest most developers will come to the LC3 specification is the parameters which they use to configure it in their applications. These are a subset of the features of Table 5.1 and are shown in Table 5.2.

LC3 Parameter	Values
Sampling Rate	8kHz, 16kHz, 24kHz, 32kHz, 44.1kHz or 48kHz.
Bits per sample	16, 24 or 32.
Frame Size	7.5ms or 10ms (The actual size for 44.1kHz sampling is generated automatically when 7.5 or 10ms is set.)
Bytes per frame (payloads per channel)	20 to 400
Number of audio channels	One. The profile may multiplex multiple channels.

Table 5.2 LC3 configuration parameters

Section 5.4 - The LC3 codec

The bits per sample at the encoder and decoder are local settings and may be different. In most cases, they are set to 16. Whatever value is chosen, the internal modules operate at a depth of 16 bits.

For unicast streams, an Acceptor can expose which combinations of these values it supports, along with a preference for which configuration is used with a specific use case. An Initiator makes a selection from the supported values exposed by each Acceptor to configure each audio stream.

To limit these to a sensible number of combinations, BAP defines sixteen configurations for the LC3 codec to help drive interoperability. These are reproduced in part below in Table 5.3 and cover sampling frequencies of 8 kHz to 48 kHz. These can be found in BAP Tables 3.5 (Unicast Server), 3.11 (Unicast Client), 3.12 (Broadcast Source) and 3.17 (Broadcast Sink).

Codec Configuration Setting	Supported Sampling Frequency	Supported Frame Duration	Supported Octets per Codec Frame	Bitrate (kbps)
8_1	8	7.5 ms	26	27.734
8_2	8	10 ms	30	24
16_1	16	7.5 ms	30	32
16_2 ¹	16	10 ms	40	32
24_1	24	7.5 ms	45	48
24_2 ²	24	10 ms	60	48
32_1	32	7.5 ms	60	64
32_2	32	10 ms	80	64
44.1_1	44.1	7.5 ms	97	95.06
44.1_2	44.1	10 ms	130	95.55
48_1	48	7.5 ms	75	80
48_2	48	10 ms	100	80
48_3	48	7.5 ms	90	96
48_4	48	10 ms	120	96
48_5	48	7.5 ms	117	124.8
48_6	48	10 ms	155	124
¹ Mandated by BAP for Acceptors and Initiators acting as unicast Audio Sinks or Audio Sources, and Broadcast Sources.				
² Mandated by BAP for Acceptors acting as unicast Audio Sinks or Broadcast Sinks.				

Table 5.3 BAP defined Codec Configuration Settings

For Broadcast Streams where the Broadcast Source has no knowledge of the receiving devices, it has to make a unilateral decision on the LC3 parameters it will use. BAP currently mandates that every Broadcast Receiver must be capable of decoding 10ms LC3 frames encoded at 16kHz with a 40 byte SDU and 24kHz with a 60 byte SDU, so a Broadcast Source using these values knows that its audio streams can be decoded by every Bluetooth LE Audio device. In

general, 16kHz sampling should only be used for voice. 24kHz is adequate for music.

Some top level audio profiles mandate support for receiving higher quality encoded LC3 audio streams. TMAP mandates support for a range of codec configurations that employ 48kHz sampling, 7.5ms and 10ms frames and a variety of bitrates. However, a Broadcast Source with no connection to all of the receiving devices cannot know whether or not such a stream can be decoded. We'll look at the implications for broadcast design later in the chapter.

5.4.4 Packet Loss Concealment (PLC)

An annoying reality with wireless transmission is that packets get lost. For Bluetooth, which shares the 2.4GHz spectrum with Wi-Fi, baby monitors and a host of other wireless products, that's generally as a result of interference. The Bluetooth specification is one of the most robust radio standards, employing adaptive frequency hopping to try and avoid any interference, but there are still occasions when data will be lost. If the audio source is a phone or a PC which is also using Wi-Fi or has other Bluetooth peripherals, there will also be occasions when they need priority, which results in a Bluetooth LE Audio transmission being missed. We'll look at ways to mitigate these later on, but the reality is that occasionally a packet will be irretrievably lost.

Unfortunately, losing a packet in an audio stream is very noticeable, and something that annoys users. To try and conceal it, a number of techniques have evolved. Inserting silence is generally annoying, unless it happens to be preceded by a silent or very quiet moment. Repeating the previous frame may work, but becomes noticeable if there are consecutive missed frames.

To provide a better listening experience, the industry has developed a range of Packet Loss Concealment algorithms which attempt to conceal the missing audio by predicting what it was most likely to be. These work very well with voice and generally quite well with music, although if a segment with, or close to an attack transient is lost, that is difficult to conceal. The LC3 specification includes a Packet Loss Concealment algorithm which has been developed to match the LC3 codec. It is applied whenever the Bad Frame Indication flag signals a lost or corrupted frame, or when the decoder detects an internal bit error. It is recommended that it, or an alternative PLC algorithm, is always used.

5.5 LC3 latency

We looked at the basics of latency at the start of the chapter, but it's important to understand its constituent parts in more detail.

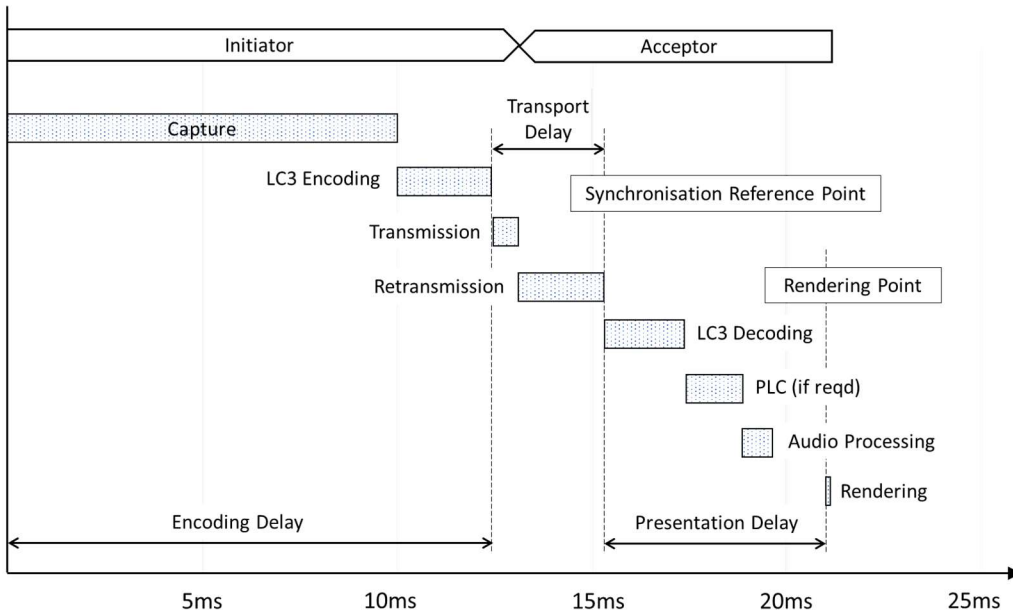


Figure 5.7 The component parts of latency

Figure 5.7 illustrates the main components of latency, showing how it is built up across the Initiator and Acceptor. Any frame-based codec starts by imposing a delay due to the sampling of the audio. For a 10ms frame length, (the standard frame length in Bluetooth LE Audio), that accounts for the first 10ms of latency. Once the incoming audio frame has been sampled, the encoding can start, which, for LC3 takes about 2.5ms, before it has a fully encoded SDU to pass forward for transmission. This includes the look-ahead elements needed for the psychoacoustic modelling.

The diagram shows transmission starting immediately. If the receiver gets a valid packet on its first transmission, the Transport Delay can be less than a millisecond, but if one or more retransmissions have been scheduled, it will take a few more milliseconds before the last possible transmission would be received at the Synchronization Reference Point. The earliest that can occur with Bluetooth LE Audio is around 14ms from the point where the first sample for that audio frame was taken, and is the fixed point in time at which every Acceptor can start to decode their received packets.

If the Initiator has decided to use a longer Isochronous Interval, or has set FT, BN or PTO parameters above 1, then the Transport Delay will increase accordingly.

The Synchronization Reference Point is where the Presentation Delay starts. Within this period, the Acceptor needs to decode the LC3 packet, which takes a few milliseconds for the

LC3 decoder, and then apply the packet loss concealment (PLC), if it detects a corrupted or lost packet, which takes another few milliseconds. Although it's not needed for most packets, the time to run the PLC algorithm has to be allocated for the occasions where it is required. If any other audio processing needs to be done, such as algorithms for noise cancellation or speech enhancement, they need to be completed before the end of the Presentation Delay, which is where each Acceptor renders the reconstituted audio stream. Because the Basic Audio Profile requires that every Acceptor must support a value of 40ms for Presentation Delay, they need to support around 40ms of buffering to hold the decoded audio data before the rendering point. In practice, the value of Presentation Delay may be lower or greater – manufacturers of receiving devices may support a range from as low as 5ms, up to several hundred milliseconds.

If everything is optimised, the quickest this whole process can happen for a 10ms LC3 packet is just under 20ms, using a very short Presentation Delay. Using a 7.5ms frame makes little difference, as the shorter frame needs a longer look-ahead delay, so the saving is only around 1 ms.

A 20ms delay is equivalent to the time it takes sound to travel about 7m. Our hearing has evolved to cope with this level of delay. If we hear an original sound and an echo 25 – 30ms later, the brain processes it without any difficulty. That means that we can use Bluetooth LE Audio Streams for earbuds and hearing aids which pick up the ambient sound as well as a Bluetooth stream without the wearer being distracted by any echo effects. However, the example above involved a lot of optimisations. It assumes that transmission can start as soon as the encoding is complete and that all retransmissions happen within a quarter of a frame, which is only valid for small packets and the lower sampling frequencies. In most real applications other factors come into play, which brings us to the Quality of Service or QoS.

5.6 Quality of Service (QoS)

Quality of Service is a term applied to the received audio signal and encompasses latency, the perceived sound of the decoded audio and the incidence of any audio artefacts, such as pops, crackles and gaps. All of these features have trade-offs with each other. The latency example described above is a highly idealised one, where packets arrive when they're expected. In practice, they don't. The human body is very efficient at absorbing Bluetooth signals, so if someone is wearing earbuds, but has their phone in the back pocket of their jeans, the signal between the phone and the earbuds may be attenuated by up to 80dB. If you're in a room, that may not matter, as the earbud will probably pick up reflected signals from the walls or ceiling. But if you're outside, where you don't have those reflecting surfaces, far more packets will be lost.

In the previous chapter, we saw that the design of Isochronous Channels adds robustness by using retransmissions, pretransmissions, burst numbers, flush timeouts and frequency hopping. If we apply enough of these features, we have a very high confidence that almost every packet will get through, and the small number that don't arrive intact can be filled in using PLC. However, as we apply these techniques, latency starts to increase, as does power

Section 5.6 - Quality of Service (QoS)

consumption. Spreading retransmissions across more than a single Isochronous Interval adds an extra frame time to the latency for each additional Isochronous Interval. Putting more retransmissions within a single frame limits the number of different streams which can be accommodated and pushes up the power – both for the transmitter, and also for the receiver, which needs to stay active to look for consecutive transmission slots.

These robustness features are separate from the codec settings. But the codec settings also have an effect on robustness. Higher quality encoding, with 48kHz sampling, will produce larger packets, which will be more susceptible to interference. Similarly, if you encode multiple Bluetooth LE Audio Streams into a single packet, i.e., the channel allocation is greater than 1, that SDU will contain multiple codec frames, resulting in larger packets, with the same problem.

Given the large number of possible codec and robustness configurations that are possible, BAP has defined sets of standard combinations aimed at the two major use cases – Low Latency and High Reliability, which can be found in Tables 5.2 and 6.4 of BAP. They cover both 10ms and 7.5ms frame intervals. The term High Reliability can sometimes be misconstrued. In these tables it indicates improved robustness, where latency is relaxed to allow for more retransmissions.

Low latency is interpreted as settings which will allow all of the retransmissions to fit within a single Isochronous Interval for sampling rates of 8kHz to 32kHz. The larger packets for 48kHz extend into two Isochronous Intervals, going up to four Isochronous Intervals for 44.1kHz. High reliability QoS configurations prioritise retransmission over latency, allowing retransmissions to be spread across six or more Isochronous Intervals for broadcast and ten or more for unicast. The term High Reliability is probably better interpreted as meaning High Robustness. Because of the higher number of retransmissions, it also leads to higher latency.

Table 5.4 shows the maximum number of Isochronous Intervals allowed for each sampling frequency, derived from these tables. The reason that the Low Latency 48 kHz sampled setting needs two Isochronous Intervals is to allow a sufficient number of retransmissions with the larger packets, as only a limited number fit into a single Isochronous Interval. For low sampling frequencies, the smaller packets mean that more retransmissions can be fitted into each Isochronous Interval. How many retransmissions are allocated is ultimately down to the scheduler in the Controller.

	Low Latency		High Reliability			
			Unicast		Broadcast	
Sampling Frequency	7.5ms	10ms	7.5ms	10ms	7.5ms	10ms
8 kHz	1	1	10	10	6	6
16 kHz	1	1	10	10	6	6
24 kHz	1	1	10	10	6	6
32 kHz	1	1	10	10	6	6
44.1 kHz ¹	4	4	12	9	8	6
48 kHz (80 kbps)	2	2	10	10	7	7
48 kHz (96 / 124 kbps) ²	2	2	10	10	7	7
<p>1 The 44.1 kHz figures differ because they are defined for framed PDUs. All other sampling rates are unframed.</p> <p>2 Multiple bitrates are defined in BAP for a sampling frequency of 48kHz, ranging from 80 kbps to 124 kbps.</p>						

Table 5.4 The maximum number of Isochronous Intervals allowed for BAP QoS settings

In the latency example of Figure 5.7, we saw that using LC3 with a 10ms frame and a significant degree of optimisation gave an overall latency of around 20ms. That used a Presentation Delay of just over 5ms. If low latency is important to an application, implementers need to be careful about their choice of QoS and codec configuration, as it can result in the latency increasing significantly. Table 5.5 shows the typical overall latency values which are likely to be achieved. These have been calculated using a value of 12.5ms for the LC3 to sample and encode a 10ms frame.

Sampling Frequency	Low Latency			High Reliability	
	Unicast and Broadcast			Unicast	Broadcast
	PD=10ms	PD=20ms	PD=40ms	PD=40 ms	PD=40 ms
8 kHz	32.5 ms	42.5 ms	62.5 ms	147.5 ms	112.5 ms
16 kHz	32.5 ms	42.5 ms	62.5 ms	147.5 ms	112.5 ms
24 kHz	32.5 ms	42.5 ms	62.5 ms	147.5 ms	112.5 ms
32 kHz	32.5 ms	42.5 ms	62.5 ms	147.5 ms	112.5 ms
44.1 kHz	53.5 ms	63.5 ms	83.5 ms	137.5 ms	112.5 ms
48 kHz (80 kbps)	42.5 ms	52.5 ms	72.5 ms	147.5 ms	117.5 ms
48 kHz (96/124 kbps)	42.5 ms	52.5 ms	72.5 ms	152.5 ms	117.5 ms

Table 5.5 Typical end-to-end latencies for BAP QoS settings for different values of Presentation Delay (PD)

The values in Table 5.5 are for single, mono audio streams. For stereo applications, the latency increases, as we will see. The message for implementers is to take care when choosing codec

Section 5.6 - Quality of Service (QoS)

and QoS settings.

The Quality of Service recommendations in Tables 5.2 and 6.4 of BAP, include suggestions for parameters to use in the HCI commands to set up unicast or broadcast streams. These recommendations (remember that the Controller uses some of these as guidance, not as definitive values) cover the:

- SDU Interval (which is the same as the Frame Duration, other than for 44.1kHz)
- Framing requirements (all are unframed, except for 44.1kHz, which is framed)
- Maximum_SDU_Size (which is the same as the Supported_Octets_per_Codec_Frame)
- A recommended Retransmission Number (RTN) for Low Latency and High Reliability options
- Max_Transport_Latency for Low Latency and High Reliability options, and
- Presentation Delay, requiring a value of 40ms to be in the supported range.

Using the values from these tables may not always produce the expected latencies. One of the reasons for that is the value for Presentation Delay. BAP requires all Audio Sinks to include a Presentation Delay of 40ms within their range of supported values, but it should not be taken as the default value – it is what it says in the note at the bottom of the table – a value that must be supported by every Acceptor acting as an Audio Sink. It's a compromise for interoperability to ensure that everything has time to decode the audio data, apply PLC and any additional processing. Most devices will be able to do better. The 40ms value in the tables represents the worst case that was expected in prototypes when BAP was written back in 2020. Since then, higher layer profiles have required tighter performance. TMAP and HAP both require Acceptors to support a Presentation Delay of 20ms for broadcast, but if an Initiator sets it to 40ms, that impacts latency. GMAP imposes overall latency requirements which require similar values. Most current devices can support values of around 10ms, although some complex audio processing algorithms may need longer.

Recalling Figure 5.7, the Presentation Delay in that example is only around 5ms, leading to an overall latency below 25ms. Many devices will be capable of supporting that, but it is highly optimised. In unicast, where the Initiator and Acceptor talk to each other, they can agree on a lower value for Presentation Delay when the Initiator is aware that it is delivering a low latency use case.

In contrast, a Broadcast Source has no way of knowing the capabilities of the Broadcast Sinks around it, so will have to make a decision based on the use case and a knowledge of the capabilities of Broadcast Sinks which are on the market and which its designers expect will access it. Both HAP and TMAP require compliant device to be able to support a Presentation Delay of 20ms, so that is a safe choice. If the lowest possible latency is required, a Broadcast Source can include an additional LTV structure, which is the Broadcast Audio Immediate Rendering Flag, which is defined in the Assigned Numbers Document. This tells an Acceptor

that it can ignore the Presentation Delay value in BASE and render the audio stream as soon as it wants. When it decides to render it is entirely up to the implementation. If it is a member of a Coordinated Set, then the two devices need some way to inform each other of what local value of Presentation Delay they will use, to ensure that the rendered audio is synchronized. How they do that is outside the scope of the Bluetooth LE Audio specifications and left to implementation.

Returning to the Low Latency columns in Table 5.5, sampling frequencies above 32 kHz risk echo effects when they are used for ambient sound applications. None of the High Reliability settings are really suitable when reinforcing ambient sound, particularly for broadcast. Few users would be able to notice the difference in audio quality between 32kHz and 48kHz sampling in a live environment, but they would notice the difference in latency.

Where an ambient audio stream cannot be heard, which is the case with most streaming music and telephony applications, latency becomes far less of a problem, unless there is an accompanying video stream, where it could lead to lip-synch problems. However, in these cases, the video application generally manages the audio stream, so can adjust the relative timing to prevent any issue.

RTN – the Retransmission Number, benefits from some further explanation. The large values in some of the configurations suggest lots of retransmissions, but that is not necessarily the case. RTN is defined in the Core [Vol 4, Part E, Sect 7.8.97] as the number of times that a CIS Data PDU should be retransmitted from the Central to the Peripheral or Peripheral to Central before it is flushed. (It can be different for the two directions.) For Broadcast, it is simply the number of times the PDU is retransmitted [Vol 4, Part E, Sect 7.8.103]. As we've already seen, the Host is not allowed to specify values for FT, PTO, NSE or BN, as it doesn't know what other constraints the Controller might have. Hence RTN, is just a recommendation to help guide the Controller's scheduling algorithm.

Most of the time, unicast audio data won't be transmitted RTN times. If an SDU has an FT greater than 1 and the SDU is transmitted the maximum number of $(RTN + 1)$ times, the following SDU will only have 1 opportunity for transmission, with no retransmissions possible. RTN gives the opportunity to get more transmission slots to accommodate short bursts of interference, but setting it too high can penalise subsequent SDUs. The average number of transmission opportunities is always NSE/BN . Having pointed that out, the values given in Tables 5.3 and 6.4 of BAP have been well tested and should generally be followed. However, as Table 6.5 of BAP shows, the Controller may choose other values.

Returning to the Isochronous Channel settings, it's useful to look at what the Initiator's Host asks for and what it actually gets. BAP gives an example of how a Controller might interpret the HCI parameters it receives based on different resource constraints. To understand that effect, we can look at the High Reliability setting for the 48_2_2 broadcast Audio Stream QoS configuration. That's defined in Table 6.4 of BAP and is reproduced in Table 5.6 below, where a maximum transport latency of 65ms is requested.

Section 5.6 - Quality of Service (QoS)

	SDU Interval (µs)	Max SDU (octets)	RTN	Max Transport Latency (ms)	Presentation Delay (µs)
48_2_2	10,000	100	4	65	40,000

Table 5.6 Broadcast Audio Stream Configuration setting for 48_2_2 High Reliability audio data

Table 6.5 of BAP provides recommended Link Layer parameters for a Controller to use when it receives an LE_Set_CIG_Parameters HCI command containing the values of Table 5.6. These are shown in Table 5.7, which also shows the resulting transport latency and the percentage of available airtime which is used for each set of parameters.

Option	ISO_Interval (ms)	BN	NSE	IRC	PTO	Num_BIS	RTN	Max_Transport_Latency Mono/Stereo	Airtime Usage Mono/Stereo
1	30	3	9	2	1	1 or 2	2	65 / 71 ms	18 / 36%
2	10	1	5	1	1	1 or 2	4	43 / 46 ms	30 / 59%
3	20	2	8	2	1	1 or 2	3	65 / 70 ms	24 / 48%

Table 5.7 Recommended BAP LL parameters for the 48_2_2 broadcast Audio Stream QoS configuration

The three options in Table 5.7 are possible because the Max_Transport_Latency and RTN are only recommendations to guide the Controller when it works out its scheduling. Depending on what else it is doing, it will normally need to take other constraints into account. The three options in Table 5.7 are recommended Link Layer settings for the following three cases:

- **Option 1**, which has the lowest airtime, is optimised for coexistence with Wi-Fi and other Bluetooth devices operating at 7.5ms schedules. Using a larger Isochronous Interval, to carry multiple 10ms frames, provides the largest gaps for Wi-Fi operation. If the Broadcast Source is a phone or a PC, and is using Wi-Fi to obtain the music stream to send over Bluetooth LE Audio, that’s very important. This option uses the least airtime for the Bluetooth LE Audio transmissions, but has the highest latency.
- **Option 2** is designed to provide the highest reliability and lowest latency, maximising the number of retransmissions in each frame. It uses the greatest amount of airtime, so is only really suitable for broadcast devices which have no other 2.4GHz radio operations.
- **Option 3** aims for a balanced approach between reliability and coexistence. It would be a good choice for a Broadcast Source which is using Wi-Fi to obtain the music stream, but which has no other coexistence issues to contend with.

These are only three of many possible combinations which can be chosen by a chip’s scheduler. Over time, others may be added to the recommended list, as the industry acquires more experience with Bluetooth LE Audio.

Before leaving this subject, Table 5.8 shows the overall latencies for stereo broadcast streams using each of these three options, with 20ms Presentation Delay mandated by TMAP and

HAP, and the baseline 40ms of BAP. The Controller will inform the Host of which parameters it has chosen, but the Host has no control over what that choice will be. That will be down to the scheduling algorithm in the chip. Designers should be aware of this variation. If the latency for your application is critical, ask your silicon manufacturer for details of what choices they are making in their Controller scheduling.

	Overall Latency	
	PD=20ms	PD=40ms
Option 1	122.3 ms	142.2 ms
Option 2	78.4 ms	98.4 ms
Option 3	112.0 ms	132.0 ms

Table 5.8 Overall latency for a broadcast stereo stream using the BAP LL options for the 48_2_2 QoS configuration

5.6.1 Airtime and retransmissions

We touched on airtime in Table 5.7. Although the LC3 codec is more efficient than previous Bluetooth codecs, as the bitrate is increased, the packets take up an increasing amount of the available airtime. The figures in Table 5.7 for Option 2 show that a stereo stream using these parameters accounts for almost 60% of the available airtime. That doesn't include the airtime required for advertising, other active Bluetooth links or any other 2.4GHz radio activity.

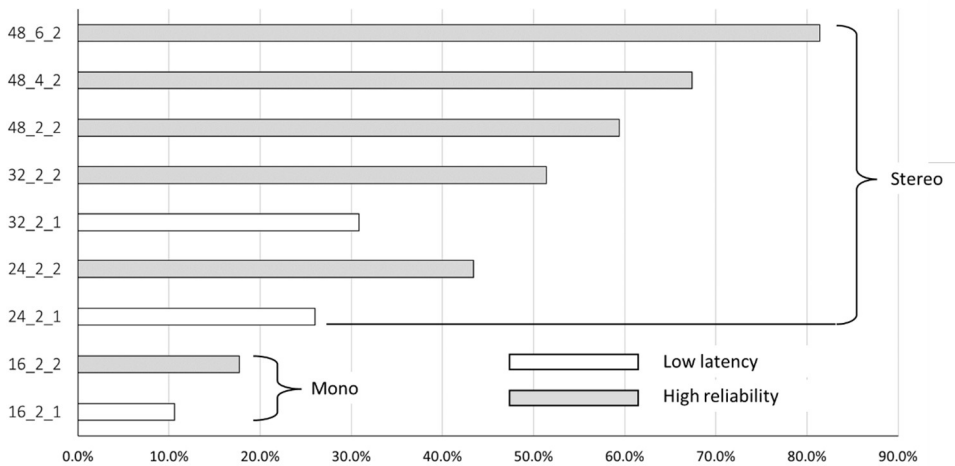


Figure 5.8 Airtime usage for different unidirectional QoS configurations

Figure 5.8 shows the effect of the higher bitrates in the QoS configurations (using a 10ms frame size), as well as the impact of more retransmissions specified by the High Reliability settings. For the 48kHz sampling configurations, the airtime is the same for Low Latency and High Reliability, as a minimum retransmission number of 4 is recommended for each because of the greater susceptibility of the larger packets to interference.

Section 5.7 - Audio quality

Broadcast Sources have no idea of whether their retransmissions have been received, so have to transmit every packet. Unicast Initiators only need to retransmit packets if they fail to receive an acknowledgement. That means that in many cases a unicast Initiator will transmit far fewer packets, providing additional airtime for any other resource on the device which needs it. However, if the link budget for the connection is poor, as it may be for devices like earbuds and hearing aids, particularly when used outside, they may need to transmit the maximum number of retransmissions, so the airtime must be allocated for this eventuality.

Airtime is a limited resource, and increasing the audio quality and reliability of a stream eats into it. One of the design requirements for Bluetooth LE Audio was the ability to support multiple Bluetooth LE Audio streams, allowing a TV or cinema to transmit soundtracks in multiple languages. Looking at Figure 5.8, it's clear that is not possible with any of the 48kHz sampling configurations, unless multiple radios are employed to transmit each different stereo language stream. In contrast, a Low Latency 24kHz or 32kHz configuration could easily cope with two different stereo streams. If product designers want to take advantage of Bluetooth LE Audio's ability to transmit multiple streams, they need to consider the airtime implications. Which brings us to audio quality.

5.7 Audio quality

Since the early days of electronic audio reproduction, a small group of users have constantly pressed for higher quality. That led to technical advances in recording and reproduction, along with marketing of terms like Hi-Fi. As well as real technical advances, it saw the appearance of pseudo-scientific fashions such as oxygen free copper cables and gold-plated connectors to “ensure” that the analogue signals were not degraded. Digital technology didn't lessen the enthusiasm for these, despite the fact that a gold-plated USB connector is an anachronism. Audio devotees kept clamouring for enhanced quality, with the digital age seeing calls for even higher sampling rates, lossless codecs, increased output levels and even a return to valve⁴⁶ amplifiers. The fact that many people over 30 now have a level of hearing loss which means they are incapable of hearing any of these “improvements” doesn't stop product marketing managers pushing for ever higher audio quality.

In its early days, Bluetooth audio attracted a fair amount of negative coverage. Some was well-deserved, as some A2DP headsets had resource constrained codec implementations. The transducers used for rendering audio were also relatively primitive. Much has changed since then, with massive development in headphones, earbuds and speakers, to the point where few users have concerns over Bluetooth technology as an audio solution and it is routinely used in top-end audio equipment costing thousands of dollars.

During the LC3 codec development, the Bluetooth SIG commissioned independent research on its audio quality compared with other codecs. The results confirm that it offers equivalent

⁴⁶ Vacuum Tubes for American readers.

or better subjective performance to other codecs, across the entire spectrum from 8kHz to 48kHz sampling. Examples of LC3 encoded audio streams are available to listen to at the Bluetooth SIG website⁴⁷, and the results of subjective listening tests compiled in an LC3 White Paper.

The tests were performed by a bank of expert listeners, using high quality, wired headphones in audio listening booths. They were asked to rate each sample of sound against the reference recording, grading how close they felt it was to the original. The tests used the MUSHRA⁴⁸ protocol, with a mixture of sounds from the EBU's⁴⁹ test recordings.

It is always difficult to relate test results like these, which are done in perfect listening conditions, with the real-world experience, because they are subjective. However, I have tried to describe the general application for the different LC3 sampling frequencies in Table 5.9.

Sampling Rate	Description
8 kHz	Suitable for basic telephony quality voice.
16 kHz	Higher quality voice. Adequate for voice recognition applications.
24 kHz	Adequate for music where there are less than perfect listening conditions, such as background noise, or where a listener has any hearing impairment. Listeners are likely to detect a difference from higher sampling rates if they are concentrating on the audio stream in noise-free surroundings.
32 kHz	Most users will not detect a difference from the original when listening with any background noise.
48 kHz	Users cannot detect a difference to the original ⁵⁰ .

Table 5.9 A subjective description of LC3 audio quality for different sampling rates.

What is important is that audio application designers understand that there are compromises in designing a wireless audio experience and that choosing the highest QoS options may exclude the development of new use cases. Some sections of the audio industry assume that they have to use the higher quality that LC3 offers and promote the highest sampling rates, despite the fact that limitations in reproduction and the listening environment will probably

⁴⁷ <https://www.bluetooth.com/blog/a-technical-overview-of-lc3/>, which includes a link to an audio demo.

⁴⁸ Multiple Stimuli with Hidden Reference and Anchor methodology for testing perceived quality, defined in ITU BS.1543-3.

⁴⁹ European Broadcasting Union TECH 3253 - Sound Quality Assessment Material recordings for subjective tests

⁵⁰ The use of 48kHz will probably be driven predominantly by marketing considerations, as few listeners can hear the difference. In listening tests, the majority of users thought that audio tracks encoded at 48kHz sounded better than the originals. Which raises interesting questions about perceived sound quality.

Section 5.8 - Multi-channel LC3 audio and Audio Configurations

mean that few, if any listeners will appreciate them. In addition, the resulting latency may be unsuitable for live applications and some devices may not be able to decode them. Others will look at the new features and use cases that are enabled by Bluetooth LE Audio and choose 24kHz or 32kHz as an acceptable compromise which allows new use cases to develop. Only time will determine what users value most. It may be more flexibility in their applications, or a desire to see a bigger “quality” number in the marketing literature.

In the past, there have been two occasions when the audio industry took a decision to reduce audio quality. The first was the introduction of CDs. The second was the use of MP3, which enabled audio streaming. Each time, there was a balance between greater ease of use versus maintaining the current audio quality. On both occasions, consumers expressed an overwhelming preference for ease of use over audio quality, despite howls of protest from a tiny minority of audiophiles. We may see this repeated with the roll-out of the Auracast™ experience.

5.8 Multi-channel LC3 audio and Audio Configurations

In Chapter 3, we introduced the concept of Audio Channel Allocation and multiplexing. It means that there are multiple different ways of transmitting the same audio data between devices. To understand how they work, it’s best to look at a couple of examples. In each of these, the following nomenclature is used to describe how audio channels are multiplexed into a CIS or BIS. In the diagrams in this section, the number of arrowheads on each CIS or BIS corresponds to the number of audio channels that it is carrying. To be absolutely clear, the double arrow indicates a CIS carrying packets with two codec frames, for example left and right.

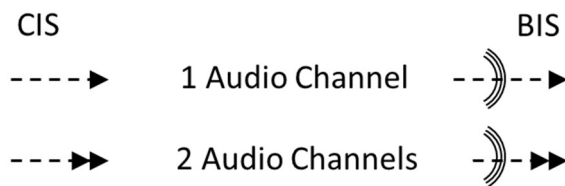


Figure 5.9 Representation of the number of Audio Channels in a CIS or BIS

Before we start, there’s one other new feature that we need to introduce, which it Audio Configurations.

5.8.1 Audio Configurations

The topology of Bluetooth LE Audio is extremely flexible, with both Initiators and Acceptors able to support multiple Isochronous Channels, each containing one or more⁵¹ encoded audio

⁵¹ If it is more than one, the audio channels need to be multiplexed.

channels. For unicast applications, an Initiator should be able to work out what an Acceptor can support and determine how best to set up its connections. However, that runs a risk that designers of Initiators and Acceptors may make conflicting implementation decisions which could lead to incompatibilities. To try to avoid this, BAP has defined 14 common Audio Configurations between an Initiator and one or more Acceptors. It mandates support for some of them, and applies further conditional and optional requirements on the others.

Table 5.10 shows the requirements for connections between an Initiator and a single Acceptor. “M” means that they are mandatory and must be supported by Bluetooth LE Audio compliant devices. “C” is a conditional support, which is generally based on whether a device supports multiplexed or bidirectional CISEs. “O” means that support is optional. Full details of these, including what the conditional requirements are, can be found in Tables 4.2 and 4.24 of BAP. TMAP and GMAP introduce and mandate support for additional Audio Configurations.

The purpose of Audio Configurations is to allow higher level profiles to raise the bar for what Initiators and Acceptors have to support, ensuring that they are capable of supporting more complex use cases. That does not mean that these configurations have to be used, but Initiators qualifying to these profiles must be able to work in the mandated Unicast Client Audio Configuration if requested by their host application, and Unicast Servers must be able to do the same if requested by a Unicast Client.

Audio Configurations 6 to 9, and 11 involve two CISEs in order for an Initiator to support two Acceptors. In Table 5.10, they bear the suffix (i), indicating that they refer to the use case where a single Acceptor is involved for each stream, with the Initiator supporting two CISEs.

Audio Configuration	Stream Direction (Initiator – Acceptor)	Unicast		Broadcast	
		Initiator	Acceptor	Initiator	Acceptor
1	----->	M	M¹	Not Applicable	
2	<-----	M	M²		
3	<----->	C	C		
4	----->>	O	C		
5	<----->>	C	C		
6 (i)	-----> ----->	M	C		
7 (i)	-----> <-----	C	C		
8 (i)	-----> <----->	C	C		
9 (i)	<----- <-----	M	C		

Section 5.8 - Multi-channel LC3 audio and Audio Configurations

Audio Configuration	Stream Direction (Initiator – Acceptor)	Unicast		Broadcast	
		Initiator	Acceptor	Initiator	Acceptor
10	<<-----	O	C		
11 (i)	<-----> <----->	C	C		
12		Not Applicable		M	M
13				M	C
14				O	C
Notes: 1 – Mandatory if the Acceptor is an Audio Sink 2 – Mandatory if the Acceptor is an Audio Source					

Table 5.10 Audio Configuration requirements for use cases with single Acceptors, denoted as (i)

Note that a Broadcast Transmitter has no way of knowing what an Acceptor supports. For this reason, Audio Configuration 14, which transmits two multiplexed, encoded audio channels should be used with care, as Acceptors do not need to support it. It is not recommended for any public broadcast application.

Table 5.11 shows the requirements where the Initiator establishes the CISes with a pair of Acceptors, with one CIS connected to each. These are denoted by a suffix (ii).

Audio Configuration	Stream Direction (Initiator – Acceptor)	Unicast		Broadcast
		Initiator	Acceptor	
6 (ii)	-----> ----->	M	See below	Not Applicable
7 (ii)	-----> <-----	C		
8 (ii)	-----> <----->	C		
9 (ii)	<----- <-----	M		
11 (ii)	<-----> <----->	C		

Table 5.11 Stream configuration requirements for sets of two Acceptors denoted as (ii)

In Table 5.11 there are no entries for the unicast Acceptor. That's because Acceptors do not need to be aware of the presence of any other Acceptor. In terms of how they are configured to accept a CIS, they act independently. Therefore, in terms of an Audio Configuration, they behave as independent devices, being configured in the Audio Configurations of Table 5.11 as supporting Audio Configurations 1, 2 or 3. This is recognised in BAP 1.0.2 and above.

Many other combinations are possible, but a Bluetooth LE Audio device can only expect the ones denoted as mandatory to be present. An Initiator can determine support for other Audio Configurations from the PAC records and Additional_ASE_Parameters values in the Codec_Specific_Configurations.

Returning to the subject of multiplexed audio streams, Figure 5.10 shows the two possible options for transmitting a stereo stream between an Initiator and a single Acceptor which can render stereo audio, using Audio Configurations 2 and 6(ii). This is the simple unicast use case of streaming music from a phone to headphones or from a TV to a soundbar.

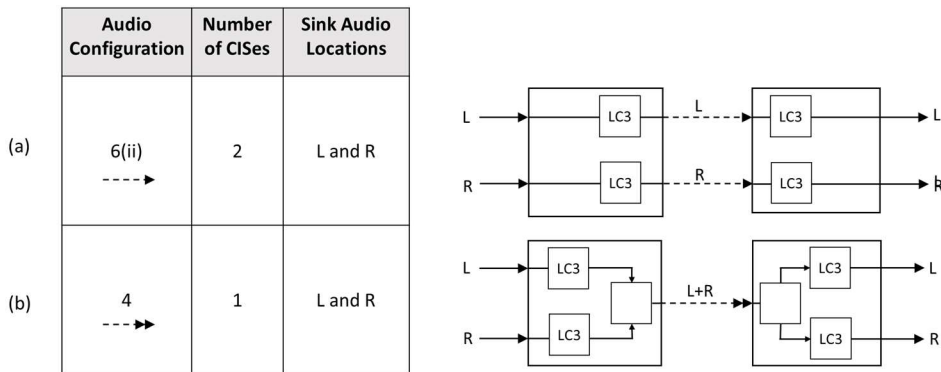


Figure 5.10 Multiplexing options for a stereo stream from an Initiator to one Acceptor

At the top of Figure 5.10, option (a) uses two separate CISes, one to carry the left audio channel and the other to carry the right channel. Below that, Option (b) sets the Channel Allocation to 2, so that both of the LC3 encoder outputs are multiplexed into a single payload by the Controller, which is sent in a single CIS. At the Acceptor, the individual left and right payloads are separated in the Controller, decoded and rendered as individual channels after the Presentation Delay. The multiplexing and demultiplexing of the encoded packets can be performed in the host or the controller. That is up to the implementation.

The example becomes a little more complex when we look at a pair of Acceptors, such as a set of left and right earbuds, as shown in Figure 5.11. In the top example (a), which is one of the mandatory schemes for Acceptors in BAP, the Initiator configures each of the Acceptors to receive a single CIS containing only the encoded audio which each one needs. As each earbud will only expose a single Sink Audio Location of Front Left or Front Right, the Initiator knows which content to send to each.

Section 5.8 - Multi-channel LC3 audio and Audio Configurations

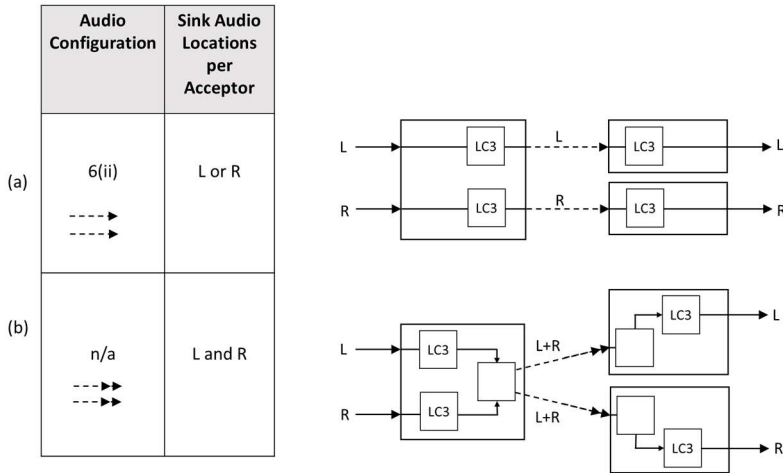


Figure 5.11 Sending a stereo stream to two earbuds

In the lower example (b), the Initiator also sets up a single CIS with each Acceptor, but in this case sends a multiplexed stereo stream to each. In order to allow this configuration, each Acceptor must expose a Sink Audio Locations characteristic value which supports both Front Left and Front Right. Each Acceptor will decode the multiplexed packet and discard the audio data which is not relevant. Note that there is no defined Audio Configuration for this arrangement, and that the ability to receive a multiplexed stream is not mandatory for an Acceptor. Hence choosing this option is not only suboptimal in terms of airtime usage, but may not be interoperable.

Sending the same unicast stereo input to a mono Acceptor is a little more interesting, as at some point between the input and output, the stereo signal needs to be downmixed to a single mono stream. There are three possible approaches to do that, which are shown in Figure 5.12.

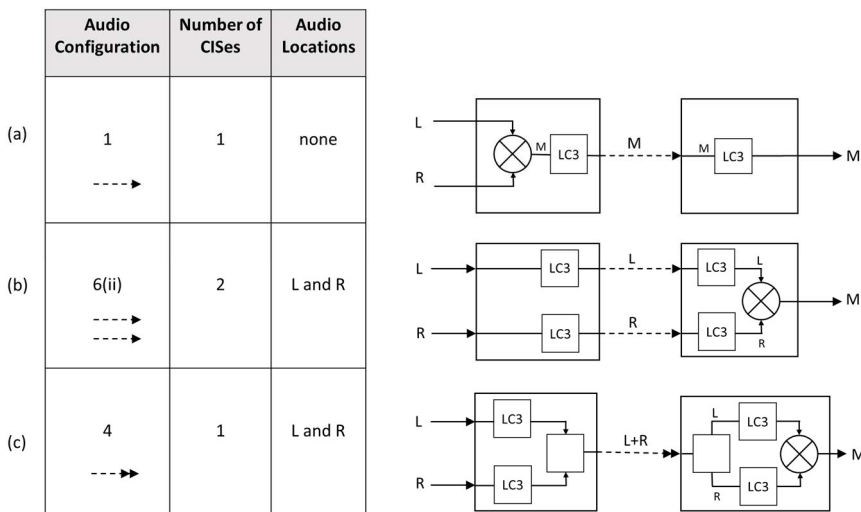


Figure 5.12 Transmitting unicast stereo to a mono Acceptor

For option (a), the Acceptor would set its Audio Sink Location as Mono⁵². That requires the Initiator to downmix its stereo input before it performs LC3 encoding, which is outside the Bluetooth LE Audio specification. This configuration gives the best use of airtime, and will be interoperable with any Acceptor acting as an Audio Sink.

For options (b) and (c), the Acceptors' Sink Audio Locations characteristic needs to be set to Front Left plus Front Right, so its Supported Sink Audio Locations value would be 0x0000000000000011. This ensures that the Acceptor will receive the two audio channels, so that it can perform the downmix. Every Initiator can support option (b), but will be unaware that its stereo stream is being rendered as mono.

In Option (b) and Option (c), the Channel Allocation and Audio Sink Location information the Acceptor exposes during the Codec Configuration procedure is identical to what it would have set if the Acceptor were a stereo device (see Figure 5.10). It means that the Initiator has no way of knowing whether it is rendering a mono or a stereo device⁵³. Therefore, it supplies it with stereo information, either as two separate CISes in Option (b), or a multiplexed stream on a single CIS in Option (c). In both cases, the Acceptor is responsible for downmixing the resultant streams. This may be configurable at the Acceptor, potentially by a simple switch, but the Initiator is unaware of it. It is worth noting that none of these three combinations are covered in the specifications. Whether an Initiator can downmix stereo to mono is out of scope of the Bluetooth specifications, and a requirement for an Acceptor to support two CISes when it can only render one stream is not mandatory. Designers of Acceptors which intend to render mono would be advised to support at least options (a) and (b). Neither Initiators nor Acceptors are mandated to support the multiplexed Audio Configuration (c), but an Initiator supporting it can determine if the Acceptor supports it, and if it cannot, can revert to option (b).

5.9 Additional codecs

The LC3 is a very good codec, which can be used across a wide range of sampling rates for voice and music applications. It is mandatory for every Bluetooth LE Audio device to support it at the 16kHz and 24kHz sampling rates (Initiators only need to support 16kHz, as some public address systems may be voice only), but the majority of implementations are likely to support higher sampling frequencies.

Despite that, there are specialised applications where other codecs may perform better. To accommodate this, the Bluetooth LE Audio specifications allow the use of additional codecs, or vendor specific codecs.

⁵² Note that supporting only mono as an Audio Location was not allowed in early versions of BAP, so this may pose additional interoperability issues.

⁵³ The Initiator could look for an instance of the Volume Offset Control Service and infer from its presence that it's a stereo device, but that may be trying to be too clever.

Section 5.9 - Additional codecs

Additional codecs used to be called optional codecs in A2DP. These are codecs which are designed by external specification bodies or companies, but the Bluetooth specifications include configuration information that allow qualified devices to recognise that they exist and how to set them up. This allows multiple manufacturers to add the capability to use them, knowing that they will work together. Without that, a connection would default back to the mandatory Bluetooth codec. Normally, additional codecs are licensable from external standards organisations or specialist companies, so anyone can integrate them into their product. Currently, no additional codecs are defined for Bluetooth LE Audio, but that will almost certainly change over time, as it did with A2DP.

Vendor specific codecs are ones which are licensed by manufacturers, who provide that Bluetooth configuration information to their licensees. Other Bluetooth products would not understand that information, so would ignore them and use the mandatory codec, or an additional codec (if they supported it). Vendor codecs are proprietary and outside the scope of the Bluetooth specifications. Where they are used, the Codec_ID is set to Vendor Specific.

Chapter 6. CAP and CSIPS

Within the Generic Audio Framework of the Bluetooth® LE Audio set of specifications, the four BAPS specifications (the Basic Audio Profile, the Audio Stream Control Service, the Broadcast Audio Scan Service and the Published Audio Capabilities Service) do most of the heavy lifting. If you implement these four specifications, you can build almost any unicast or broadcast application. However, they are designed to be as generic as possible, which means that there are multiple, different ways of putting the pieces together. CAP – the Common Audio Profile, defines a set of procedures to establish common ways to perform all of the everyday actions that are needed to set up Audio Streams between an Initiator and one or more Acceptors. For most developers, CAP provides the interface on which they build their applications.

CAP ties in the content control, use case and rendering control concepts which we came across in Chapter 3. It defines when these need to be used during the stream configuration and establishment process. It is also key to preventing multi-profile issues when a Bluetooth LE Audio device transitions between different use cases within a connection, or makes connections with different devices.

The other thing that CAP brings to the process is coordination. The biggest market for Bluetooth audio today is in earbuds – two separate devices which need to work as if they were one. The BAPS specifications deal with individual streams. CAP lays down rules for managing streams when an Initiator connects to multiple Acceptors, using the Coordinated Set Identification Profile and Service to bind them together.

Finally, CAP defines the Commander role, which is what elevates broadcast from a simple telecoil replacement to a highly flexible Bluetooth Audio distribution solution.

In this chapter, I'll provide an overview of the CAP procedures and how they're used. Essentially, you can think of CAP as a recipe book. The BAPS specifications define all of the ingredients for Audio Streams and CAP tells you which ones to use for each procedure, and what order to use them in. Most of the detail will come out in the following chapters on setting up unicast and broadcast Audio Streams, where we'll see how CAP augments and utilises the stream control and management procedures that are already in BAPS. But before we start on CAP, it's important to understand CSIP and CSIS.

6.1 CSIPS – the Coordinated Set Identification Profile and Service

Because A2DP was designed for streaming to a single device, every Bluetooth solution on the market today that sends audio to multiple devices uses some proprietary method of making them work together, whether they're a pair of earbuds, hearing aids or speakers. Bluetooth LE Audio needed to define a standard method of doing this, so that any combination of products could be used together. The issue is not just making sure that they can have their volume controlled together, but also to make sure that if you change the device providing the

audio stream, such as when a phone call interrupts you when you're watching TV, then both left and right earbuds change their connection to your phone at the same time. In Bluetooth LE Audio, the requirement was to ensure that two separate devices could act in perfect unity, even if there was no connection between them. That led to the concept of Coordinated Sets.

The Coordinated Set Identification Service is instantiated on devices which form a group (called a Coordinated Set), where the group members fulfil a specific use case, acting in a concerted manner. A typical example is a pair of earbuds. The specifications are not limited to audio devices – they could equally be used for sets of medical sensors, such as ECG patches, where data is being collected from separate sensors. Nor do all of the functions of those devices need to be exactly the same, but one feature should be. For example, in a pair of earbuds, the common feature is that they would both render audio streams, but only one needs to have a microphone.

For a pair of earbuds, coordination performs four main functions:

- It identifies the members of the Coordinated Set, i.e., a left and right earbud
- It is used to apply volume controls and mute to both devices
- It is used to ensure they both receive audio streams from the same Audio Source (the streams themselves are generally different, being left and right, but that is irrelevant to CSIPS, and
- It allows a device to lock access to the earbuds, preventing other devices from accessing them during a control operation.

Note that CSIPS is not responsible for the accurate rendering of audio from left and right devices. That's controlled by the Presentation Delay.

CSIP defines two roles:

- A device which is a member of a Coordinated Set is a Set Member, and
- A device which discovers and manages a Coordinated Set is a Set Coordinator.

The Set Member role is essentially a passive role – it simply states that it is part of a set. The Set Coordinator not only finds the members, but then passes that knowledge on to other procedures to ensure that those procedures act on all members of the set.

CSIS requires that every member of a Coordinated Set includes a Resolvable Set Identity (RSI) AD Type, which allows a Client device to recognize that it is a member of a Coordinated Set. This tells the Client that there are two or more Acceptors that need to be found. The RSI is a random six-octet identifier, which changes over time. This reduces the risk of someone tracking your Bluetooth products.

Section 6.1 - CSIPS – the Coordinated Set Identification Profile and Service

Once a Client device, which can be an Initiator or Commander, makes a connection to a device exposing the RSI AD Type, it pairs and bonds with the set member it has discovered and reads its Set Identity Resolving Key (SIRK) characteristic. The SIRK is a 128 bit random number which is common to all members of the Coordinated Set, which a Client can use to decode the Resolvable Set Identity. It does not change for the lifetime of the device⁵⁴. The SIRK is normally programmed into all of the devices which comprise the Coordinated Set during manufacture. The Bluetooth LE Audio specifications do not specify a way to set it or change it, so if products need to be added to a Coordinated Set during their lifetime, such as when a lost or broken earbud is replaced, manufacturers will need to determine a method to accomplish this.

The Client also needs to read the Coordinated Set Size characteristic, which tells it how many devices there are in that Coordinated Set. It then proceeds to find the other members of the set by using the Set Members Discovery Procedure defined in CSIP. This involves the Client device looking for other Acceptors exposing the RSI AD Type, connecting and pairing to them and reading their SIRK characteristic. If the SIRK has the same value as the first coordinated device, then it is a member of the same set. If it is different, the Client device should discard the pairing and look for the next device with an RSI AD Type and check its SIRK characteristic value. The Set Members Discovery process ends when all of the set members are found, the process reaches an implementation-specific timeout (usually around ten seconds), or it is terminated by the application. If the Initiator or Commander can't find them all, they can proceed with those they have found, but should continue to try to find the missing members.

CAP demands that if you are shipping Acceptors which are part of a Coordinated Set, then all four of the characteristics defined in CSIS are implemented in each device. These are shown in Table 6.1.

Characteristic Name	Mandatory Properties	Optional Properties
Set Identity Resolving Key (SIRK)	Read	Notify
Coordinated Set Size	Read	Notify
Set Member Lock	Read, Write, Notify	None
Set Member Rank	Read	Notify

Table 6.1 Coordinated Set characteristic properties for use with CAP

Currently, no CAP procedure uses the Set Member Lock or Rank characteristics, although it mandates that they are implemented on the Acceptor.

The reason for the Set Member Lock is that when a Client writes that characteristic, the Lock

⁵⁴ A firmware update is considered to be the start of a new life.

gives that Client exclusive access to features on that Acceptor. Which features the Lock controls is defined by the higher-layer application. Implementors should take care in using the Lock, as it can prevent other Commanders or Initiators accessing the Acceptors. When a Client no longer needs exclusive access, it should release the Lock.

The Rank is a value that is normally assigned at manufacture to provide a unique number to each member of the Coordinated Set. It doesn't imply any priority, but is a unique, positive integer within the Coordinated Set which is used to ensure that all Clients apply their operations to the members of the set in the same order. Rank is used in the Ordered Access procedure in CSIP. This states that when applying a Lock for any reason, Clients should start with the set member that it knows to have the lowest rank and then work up through the other available members of the Coordinated Set in ascending numeric order of Rank. That prevents a race condition where two different Clients apply a Lock to different set members at the same time. The Rank values are normally set during manufacture.

The Ordered Access procedure is also used by CAP as a precursor to running any CAP procedure. It checks whether any Acceptor is locked and only continues with the CAP procedure once it has determined that none of the set members have a Lock set. It then applies the CAP procedure to each Acceptor in order of increasing Rank.

6.2 CAP – the Common Audio Profile

As I said above, CAP can be considered as the recipe book for all of Bluetooth LE Audio, bringing together all of the other specifications into five main sets of procedures which define the way that everything works. Top level profiles, like HAP, TMAP, GMAP and PBP then refer to these CAP procedures, adding additional requirements for their specific use cases.

CAP is where the Initiator, Acceptor and Commander roles that I'm using throughout this book are defined. Whilst only the Initiator and Acceptor can be involved in Audio Streams, CAP describes how all three of these roles can include a range of components from the other Generic Audio Framework specifications. These are listed in the matrix of Table 6.2, which shows Mandatory, Optional, Conditional and Excluded requirements. Conditional features are normally mandated for specific combinations of other features, and require that a role supports at least one of a number of optional components. Excluded combinations are not allowed. The full details of all of these combinations can be found in Table 3.1 of CAP. Boxes with a dash (-) are features which could be implemented in a device, but are outside the scope of the CAP procedures.

Section 6.2 - CAP – the Common Audio Profile

Component \ Role	Acceptor	Initiator	Commander
BAP Unicast Client	X	C	X
BAP Unicast Server	C	X	X
BAP Broadcast Source	X	C	X
BAP Broadcast Sink	C	X	X
BAP Broadcast Assistant	X	X	C
BAP Scan Delegator	C	X	C
VCP Volume Controller	X	-	C
VCP Volume Renderer	O	X	X
MICP Microphone Controller	X	-	C
MICP Microphone Device	O	X	X
CCP Call Control Server	X	O	X
CCP Call Control Client	O	X	-
MCP Media Control Server	X	O	X
MCP Media Control Client	O	X	C
CSIP Set Coordinator	X	C	C
CSIP Set Member	C	X	X
“O” = Optional, “C” = Conditional, X = Excluded, “-” = Not relevant to this profile.			

Table 6.2 Component support requirements for CAP roles. See Table 3.1 of CAP for individual conditions

6.2.1 CAP procedures for unicast stream management

The CAP procedures for managing streams can be grouped into three categories. The first is a set of three procedures for unicast streams, which are largely self-explanatory:

- Unicast Audio Start procedure
- Unicast Audio Update procedure
- Unicast Audio Stop procedure

The Unicast Audio Update procedure allows certain features of the audio stream to be changed, predominantly through metadata updates. It does not allow the basic CIS parameters to be changed. Once they have been set by the controller, they are fixed for the life of the CIG.

These procedures involve both the Initiator and the Acceptor, as setting up each unicast stream uses command and response sequences.

6.2.2 CAP procedures for broadcast stream transmission

The second set of three procedures is for broadcast streams:

- Broadcast Audio Start procedure
- Broadcast Audio Update procedure
- Broadcast Audio Stop procedure

The Broadcast Audio Start, Stop and Update procedures are only used by the Initiator, which is set up unilaterally, with no knowledge of whether any Acceptors are present. As with the CIS Update, it is only metadata which can be updated during the BIG's lifetime.

6.2.3 CAP procedures for broadcast stream reception

To complement the broadcast procedures for the Initiator, the third set of stream management procedures is for Acceptors that want to acquire the broadcast Audio Streams, giving us two procedures:

- Broadcast Audio Reception Start procedure
- Broadcast Audio Reception Ending procedure

There is no broadcast Audio Update procedure for the Acceptor, as an Acceptor is a passive entity in broadcast which only consumes the Audio Stream. It cannot do anything to update the content of the broadcast streams, hence there is no update procedure for broadcast audio reception.

6.2.4 CAP procedures for stream handover

CAP includes two procedures specifically for a stream handover, where an Initiator wants to transition between transmitting unicast and broadcast Audio Streams (and vice versa). These are the:

- Unicast to Broadcast Audio Handover procedure, and the
- Broadcast to Unicast Audio Handover procedure

These are very important, as they describe the audio sharing use case, where a user can transition from listening to a private stream from their phone or music source, to sharing it with friends by converting it to broadcast. Although the transmission topology changes between CIG to BIG, the original Initiator retains control of the Audio Stream and the user's Acceptors.

The reason for the importance of the handover procedures is that most Initiators and Acceptors do not have the resources to handle concurrent unicast and broadcast streams. For the High Reliability QoS settings, there simply isn't enough airtime to do both. That means that an Initiator usually needs to stop its unicast stream before starting the broadcast stream,

Section 6.2 - CAP – the Common Audio Profile

even for a 24kHz sampling rate. To ensure continuity for the primary user, i.e., the one who is sharing their audio with their friends, the Initiator has to apply the same Context Types to the new stream. In most cases, the primary user with the Audio Source will want to continue to control the stream, so although the Audio Stream is now broadcast, they will keep their ACL link up and associate the new broadcast Audio Stream with the same Content Control ID (CCID) they were using for the unicast stream.

In practice, there are likely to be short gaps in transmission during this handover, which will be noticeable on the original Acceptors linked to the Initiator. Whilst implementations can try to minimise these, it may be simpler to conceal them by adding a simple announcement to the user to “please wait while we move to Audio Sharing”.

6.2.5 CAP procedures for capture and rendering control

As well as the four sets of procedures for Audio Streams, CAP also contains five Capture and Rendering Control procedures, which are also self-explanatory:

- Change Volume procedure
- Change Volume Offset procedure
- Change Volume Mute State procedure
- Microphone Mute State procedure
- Change Microphone Gain Settings procedure.

6.2.6 Other CAP procedures

In addition to the Audio Stream, and Capture and Rendering Control procedures, CAP defines a set of procedures which are used with the Audio Stream procedures listed above. The first one, which we came across in the previous section, is the Coordinated Set Member Discovery procedure, which is performed by an Initiator before it sets up a unicast stream, and by an Initiator or Commander before they adjust the volume or capture settings on a Coordinated Set. Once the members of a Coordinated Set are known, CAP always applies the CSIP Ordered Access Procedure prior to any other CAP procedure. The second is the Distribute Broadcast_Code procedure, which defines how Broadcast_Codes are distributed for encrypted, private broadcast streams.

6.2.7 CAP and BAP Announcements

With the exception of broadcast use cases, Bluetooth LE Audio uses the same peripheral connection procedures that are used for other LE applications, both for device discovery, LE ACL connection and bonding. Section 8 of BAP defines the procedures, referring to the relevant sections in the Core specification and provides recommended values to use for Scan Interval, Scan Window and Connection Interval, for quick setup and reduced power situations. These are all standard connection procedures defined in the Generic Access Profile (GAP), so I won't spend any more time on them here.

Section 8 of CAP expands on the BAP requirements with considerations for multi-bond situations, where an Acceptor is bonded with multiple Initiators (for example, a phone and a TV). It also introduces two new modes which reflect the fact that both Initiators and Acceptors can make connection decisions, in order to support the concept of the “Sink led journey” described in Chapter 3. These two modes use CAP and BAP Announcements and are the:

- Immediate Need for Audio related Peripheral (INAP) mode, and the
- Ready for Audio related Peripheral (RAP) mode.

6.2.7.1 Immediate Need for Audio related Peripheral (INAP) mode

The INAP mode is used when an Initiator or Commander needs to connect to an Acceptor, typically because of a user action, such as pressing a button, or an external action, such as an incoming phone call. If it discovers an available Acceptor, it should connect. If it discovers more than one, it should present the user with a range of available Acceptors to let them make that choice, typically by displaying a “pick list” of available Bluetooth LE Audio devices.

INAP is used when a high priority response is required, so the Initiator or Commander should use the quick setup parameter values for connections described in Section 8 of BAP, scanning at a higher rate. The Initiator should follow CAP procedures to determine whether the Acceptor that responds is a member of a Coordinated Set, and if so, discover and connect to all of the other set members.

6.2.7.2 Ready for Audio related Peripheral (RAP) mode

The opposite case to INAP is the RAP mode, where an Acceptor is looking for an Initiator. It signals this by transmitting Targeted Announcements containing a Context Type describing the use case that it wants supported. If an Initiator can support it, it should attempt to connect. When in the RAP mode, the Initiator should use the reduced power parameter values for connections, described in BAP Section 8, scanning at a lower rate. Once connected to the Acceptor which was sending Targeted Announcements, the Initiator should determine whether the Acceptor is a member of a Coordinated Set, and if so, discover and connect to all of the other set members.

Acceptors should only use Targeted Announcements for a limited period of time, when they require an immediate connection.

There is a further subtlety in Announcements, which is that they may or may not include a Context Type value. If they are used in relation to an Audio Stream, they should include the Context Type value in the form of an Available Bluetooth LE Audio Context field. These are called BAP Announcements [BAP 3.5.3]. If they are used for control purposes or Scan Delegation, they do not contain this field. They are then called CAP Announcements [CAP 8.1.1]

Section 6.2 - CAP – the Common Audio Profile

The range of Initiator responses in RAP or INAP mode to different forms of announcement is described in Table 8.4 of CAP.

6.2.8 Coping with missing set members

There will be occasions when an Initiator has read the Coordinated Set Size characteristic of a Coordinated Set member, tried to locate the other members and discovered that some are missing. This could happen at the start of setting up an Audio Stream, or during the course of a session when the Initiator detects a link loss with one of the Acceptors. It may be because one or more of them are physically missing, out of range, or their battery has died. When this occurs, the Initiator should try to find the missing member, but should not stop with the connection process, nor terminate any existing streams. Instead, it should regularly try to find the missing set member(s) and add them back by re-establishing the connection once they have been discovered.

An implementation may decide to adjust the content of a unicast Audio Stream if a member is lost. For example, if your phone had been streaming a stereo music stream to a Coordinated Set of left and right earbuds and the connection to one of them disappears, it may decide to replace the remaining stream with a down-mixed mono stream. However, this behaviour is not specified in Bluetooth LE Audio and is implementation specific, as are the retry cadence and any timeouts for attempting to find missing Coordinated Set members. This should not affect the Context Type, as the use case remains the same.

There are some applications where one or more members of a Coordinated Set will always be missing, as it may have been deliberately configured with redundant members. These will normally be handled with proprietary applications to ensure that Initiators do not spend time needlessly searching for missing devices.

-oOo-

The Common Audio Profile is probably best summed up as a profile which says, “this is how to connect” and “do this for all of the streams you’re dealing with”. As such, it pulls together all of the other Generic Audio Framework specifications, providing a common set of procedures for the top level profiles to use. As more specifications are developed within GAF, CAP will expand to include them.

We’ll come across all of these CAP procedures in the next few chapters on setting up unicast and broadcast streams and Capture and Rendering Control. Most developers will find themselves working with these procedures, rather than the underlying BAP procedures, as the CAP procedures are the ones which are referenced by the top level profiles. However, understanding the BAP procedures and the parameters used for setting up and managing streams is a useful exercise to understand how everything fits together in the Bluetooth LE Audio world. That’s what we’ll do next.

Chapter 7. Setting up Unicast Audio Streams

In this chapter we'll look at how to configure and set up unicast Audio Streams. The four main specifications which are involved in doing this are:

- PACS – the Published Audio Capabilities Service,
- ASCS – the Audio Stream Configuration Service,
- BAP – the Basic Audio Profile, and
- CAP – the Common Audio Profile.

CAP sits above the other three and defines procedures which use BAP, ASCS and PACS to configure and manage unicast Audio Streams, introducing coordination, Context Types and the Content Control ID to associate Audio Streams with use cases. Most of the time, CAP is just a set of rules which tells you the correct order to run BAP procedures and how to use Context Types and the Content Control ID with them. Top level Bluetooth® LE Audio profiles, such as HAP and TMAP, are built on top of CAP, mostly extending mandatory requirements. In this chapter I'll concentrate on the underlying BAP procedures, as they do the heavy lifting, but will reference CAP and higher layer profiles where necessary.

All of the Bluetooth LE Audio specifications are based on the GATT structure of Bluetooth LE, which has a Client-Server relationship. Higher layer specifications provide context and add control to unicast Audio Streams, but BAP, ASCS and PACS are the foundations they build on.

7.1 PACS – the Published Audio Capabilities Service

PACS is the simplest of the specifications and is essentially a statement of what an Acceptor can do. Acceptors use PACS to expose these capabilities in two characteristics – a Sink PAC characteristic if it supports the Audio Sink role and a Source PAC characteristic if it supports the Audio Source role. These contain Published Audio Compatibility records, called PAC records, which contain information about the codec and the configurations which it supports across both broadcast and unicast roles, along with other optional features. Between them, they expose the full range of capabilities of a device.

PACS can be thought of as a database of the audio capabilities of an Acceptor, which is populated at the point of manufacture and is static for the life of the product, although it may be updated through a firmware update. PACS specifies information which an Initiator obtains when it starts the process of configuring and establishing an audio stream. PACS can also be used by a Broadcast Assistant to allow it to filter the Broadcast Streams which it detects, so that a Broadcast Sink is only presented with compatible Bluetooth LE Audio Streams, but we'll cover that in the broadcast chapter.

An important concept about the information specified by PACS is that it is a statement of fact about the total capabilities of an Acceptor. PACS describes everything that an Acceptor is capable of doing. It is the first step in devices getting to know each other. After an Initiator has read these capabilities, an Acceptor will normally expose a more limited set of preferred capabilities during the process of establishing a stream, and the Initiator should use those values. However, an Acceptor should not reject any configuration an Initiator requests which is based on the PACS information it has exposed.

The intent is that this should result in an efficient configuration process, especially when multiple Acceptors are involved, particularly if they come from different manufacturers and have differing capabilities. It's an evolution from the classic Bluetooth audio profiles which allow Central and Peripheral devices to go through negotiation loops, where the two devices try to ascertain what the optimum settings are for an audio connection. In some implementations, that resulted in a deadlock, with a connection never being made, or a poor codec configuration which affected the quality of the audio. It's an issue which resulted in the inclusion of the "S" and "D" coding settings as recommendations for CVSD and the "T" eSCO⁵⁵ parameter sets for mSBC. The aim of PACS is to prevent those problems arising and to streamline the configuration process.

7.1.1 Sink PAC and Source PAC characteristics

Both the Sink PAC and Source PAC characteristics contains an array of "i" PAC records, taking the form shown in Table 7.1.

Parameter	Description
Number_of_PAC_Records	Number of PAC records [i] for this characteristic
Codec_ID [i] (5 octets)	Octet 0: Codec (defined in the Bluetooth Assigned Numbers) LC3 = 0x06 Vendor Specific = xnn Octets 1&2: Company ID, if vendor specific, otherwise 0x00 Octets 3&4: Vendor specific Codec ID, otherwise 0x00
Codec_Specific_Capabilities_Length [i] (1 octet)	Length of the codec specific capabilities for the codec in the i th PAC Record.

⁵⁵ The S "safe set" parameters for CVSD were introduced in the Codec Interoperability section (5.7) of HFP v1.5 in 2005, followed by the D and T parameters for the mSBC in version 1.6 in 2011, to help ensure compatibility when using these codecs.

Section 7.1 - PACS – the Published Audio Capabilities Service

Parameter	Description
Codec_Specific_Capabilities [i] (length varies)	Identification of the capabilities of the codec implementation in the i th PAC Record, normally expressed as a bitfield.
Metadata length [i] (1 octet)	Length of the metadata associated with the i th PAC Record. 0x00 if there is none.
Metadata [i]	LTV formatted metadata for the i th PAC record

Table 7.1 Format of a PAC Record

7.1.2 Codec Specific Capabilities

Every Bluetooth LE Audio specification must include at least one PAC record using the LC3 codec, as this is mandated in BAP, which is the lowest layer of the Generic Audio Framework. The assigned number for LC3 is 0x06, so every Bluetooth LE Audio Acceptor will have at least one PAC record with the Codec_ID of 0x0000000006. (Note that the Codec_ID is the Coding_Format and Codec_ID described in the Host Controller Interface Assigned Number list (Section 2.11), and not the Audio Codec ID defined in the Audio/Video Assigned Numbers list (Section 6.5.1). These Codec_Specific_Capabilities LTVs are generic and can be used with any codec, but certain values must be supported for LC3 PAC records.

The Codec_Specific_Capabilities requirements are defined in Section 6.12.4 of the Assigned Numbers document and consist of five LTV structures. Each LTV has a Type code, so that it can be identified by the device reading it. Note that the Codec_Specific_Capabilities LTV structures in Section 6.12.4 of the Assigned Numbers document are different to the Codec_Specific_Configuration LTV structures of Section 6.2.15 and should not be confused with them. Although they are similar, some of the bit allocations are different.

As three of these LTVs are bitfields, this means that there are typically multiple combinations described within the Codec_Specific_Capabilities structure.

7.1.2.1 The Supported_Sampling_Frequencies LTV

The first of these five LTV structures is the Supported_Sampling_Frequencies (Type = 0x01, Section 6.12.4.1 of Assigned Numbers) shown in Table 7.2, which is a bitfield of sampling frequencies covering the range from 8 kHz to 384 kHz. Support for each value is indicated by setting the corresponding bit to one. BAP mandates that an Acceptor acting as an Audio Sink must always support at least 16 and 24kHz sampling frequencies, and if it is acting solely as an Audio Source, it must support at least 16kHz sampling. (Although a product must demonstrate support for these during qualification, it may never use them. They provide a lowest common denominator for interoperability, but the choice of codec configuration is entirely down to the application.)

This LTV should always be present. Note that this structure is applicable to any codec. LC3 only specifies 8, 16, 24, 32, 44.1 and 48 kHz sampling frequencies, shown as lightly shaded in Table 7.2.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
kHz	RFU			384	192	176.4	96	88.2	48	44.1	32	24	22.05	16	11.025	8

Table 7.2 Supported Sampling Frequencies [Assigned Numbers Section 6.12.4.1]

7.1.2.2 The Supported_Frame_Durations LTV

The second codec capabilities LTV structure is the Supported_Frame_Durations (Type = 0x02, Section 6.12.4.2 of Assigned Numbers) of Table 7.3, which provides information on the two frame durations supported by the LC3 codec – 7.5ms and 10ms. When set to 1, Bits 0 and 1 indicate support for the two options of 7.5ms and 10ms frames for LC3. If both 7.5 ms and 10 ms are supported, bits 4 and 5 can be used to indicate whether one of them is preferred. Only one of bits 4 and 5 can be set. This LTV is also mandatory. Support for 10ms is mandatory for all devices, so if that is the only value supported, this LTV can be omitted.

Bit	All other bits	5	4	3	2	1	0
Frame Duration	RFU	7.5ms preferred	10ms preferred	RFU	RFU	10ms supported	7.5ms supported

Table 7.3 Supported Frame Durations (Assigned Numbers Section 6.12.4.2)

7.1.2.3 The Supported_Audio_Channel_Counts LTV

The third LTV structure is the Supported_Audio_Channel_Counts (Type = 0x03, Section 6.12.4.3 of Assigned Numbers) shown in Table 7.4. This is another bitfield, which indicates the number of Audio Channels which can be included in a CIS or BIS. Channel Count is the number of multiplexed Audio Channels which can be carried in the same direction on an Isochronous Stream, which we covered in Section 5.8. Bits 0 to 7 indicate the number of channel counts that are supported, with a value of 1 representing a supported option. At least one bit must be set, otherwise it implies that no audio channel can be set up. If multiplexing is not supported, then the Channel Count is 1 and this LTV structure can be omitted. As explained in Section 3.11, the codec frames within the multiplexed media packet are arranged in order of increasing bit value for their Audio_Channel_Allocation bits.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Channel Count	RFU								8	7	6	5	4	3	2	1

Table 7.4 Supported Audio Channel Counts [Assigned Numbers Section 6.12.4.3]

7.1.2.4 The Supported_Octets_Per_Codec_Frame LTV

The fourth LTV structure is the Supported_Octets_Per_Codec_Frame (Type = 0x04, Section 6.12.4.4 of Assigned Numbers) shown in Table 7.5. This structure consists of two pairs of

Section 7.1 - PACS – the Published Audio Capabilities Service

two octets, with the lower pair (Octets 0 and 1) specifying the minimum number of encoded octets per codec frame and the upper pair the maximum number of encoded octets per codec frame. Both can be set to the same value where only a specific number of octets is supported. It defines the range of bitrates⁵⁶ supported across the selected sampling rates.

Octet	15 – 4	3	2	1	0
Value	RFU	maximum number of octets per codec frame		minimum number of octets per codec frame	

Table 7.5 Supported Octets per Codec Frame (Assigned Numbers Section 6.12.4.4)

7.1.2.5 The Supported_Max_Codec_Frames_Per_SDU LTV

The fifth and final LTV structure in the Supported_Codec_Specific_Capabilities is the Supported_Max_Codec_Frames_Per_SDU (Type = 0x05, Section 6.12.4.5 of Assigned Numbers), which is a single octet stating the maximum number of codec frames which can be packed into a single SDU. It can be omitted if there is only one frame per SDU.

7.1.2.6 The Preferred_Audio_Contexts LTV

The Codec_Specific_Capabilities can be followed by metadata, which is formatted in an LTV structure and described in the Metadata LTV structures section of the Assigned Numbers document (Section 6.12.6). Currently, the only defined metadata that is relevant for the PAC record is the Preferred_Audio_Contexts LTV. This is a 2-Octet bitfield of Context Type values, using the standard values for Context Types from the Generic Audio Assigned Numbers document. If a bit is set to 0b1, it signifies that this Context Type is a preferred use case for the codec configuration in that PAC record. Multiple bits can be set.

Normally, context types would be invoked at an application level, but there are circumstances where it makes sense to add them to a PAC record. For example, if a device wanted a specific codec to be used for particular use case, the Acceptor could include specific PAC records to tie a codec configuration to a Context Type, providing more granularity than the Supported_Audio_Contexts characteristic (q.v.).

7.1.3 Minimum PACS capabilities for an Audio Sink

The mandatory BAP LC3 requirements mean that an Acceptor which is acting as a unicast Audio Sink, i.e., receiving an Audio Stream, has to support reception of a minimum of one audio channel at 16 and 24kHz sampling frequencies with a 10ms frame duration. Similarly, every Acceptor which is acting as a unicast Audio Source, i.e., transmitting an Audio Stream, has to support encoding a minimum of one audio channel at a 16 kHz sampling frequency with a 10ms frame duration.

⁵⁶ The bitrate is eight times the number of octets.

The mandatory 16kHz sampling rate for both Audio Sink and Source requires support for 40 octets per SDU, whilst the 24 kHz sampling rate for the Audio Sink requires 60 octets. For the following example, which illustrates PAC records, we'll just look at the Sink PAC characteristic. The same principles apply to the Source PAC characteristic.

The Sink PAC characteristic would normally expose these two mandatory codec settings in a single PAC record. Alternatively, they could be separated into two PAC records, each specifying a single set of capabilities. BAP only requires support for one audio channel (Audio Configuration 1 in Table 4.2 of BAP), but to illustrate how PAC records are built, let's look at an example of an Acceptor which can support two Audio Channels (corresponding to Audio Configuration 4), but only wants to use a single channel for 16kHz QoS settings. If it exposed these as four separate PAC records, the content of each set of parameters within the Source PAC characteristic would look like Table 7.6.

Parameter for each Index	Index			
	0	1	2	3
Codec_ID (LC3)	0x000000000D			
Supported_Codec_Specific_Capabilities_Length	0x14	0x14	0x14	0x14
Supported_Codec_Specific_Capabilities				
Supported_Sampling_Frequencies	<i>(Example: 16 kHz = 0x0004, 24 kHz = 0x0010)</i>			
Length	0x03	0x03	0x03	0x03
Code	0x01	0x01	0x01	0x01
Value	0x0004	0x0004	0x0010	0x0010
Supported_Frame_Durations	<i>(Example – 10ms only)</i>			
Length	0x02	0x02	0x02	0x02
Code	0x02	0x02	0x02	0x02
Value	0x02	0x02	0x02	0x02
Supported Audio Channel Counts	<i>(Example: 1 = 0x0001; 1 and 2 = 0x0003)</i>			
Length	0x03	0x03	0x03	0x03
Code	0x03	0x03	0x03	0x03
Value	0x0001	0x0001	0x0003	0x0003
Supported Octets per Codec Frame	<i>(Example – 40 only = 0x00280028 or 60 only = 0x003C003C)</i>			
Length	0x05	0x05	0x05	0x05
Code	0x04	0x04	0x04	0x04
Value	0x00280028		0x003C003C	
Supported Max Codec Frames Per SDU	<i>(Optional. Default = 1)</i>			
Length	0x02	0x02	0x02	0x02
Code	0x05	0x05	0x05	0x05

Section 7.1 - PACS – the Published Audio Capabilities Service

Parameter for each Index	Index			
	0	1	2	3
Value	0x01	0x01	0x01	0x01
Metadata	<i>(No metadata if length is set to 0)</i>			
Length	0x00	0x00	0x00	0x00

Table 7.6 Example of the contents of a set of four Sink PAC records for mandatory 16 & 24 kHz sampling at 10ms. (Index 3 supports 1 or 2 audio channels.)

As an example of how they can be formed, Table 7.6 contains the Sink PAC record parameters for the mandated codec requirements for all supported Bluetooth LE Audio profiles i.e., the 16_2_1 (index 1), 16_2_2 (index 2), 24_2_1 (index 3) and 24_2_2 (index 4) QoS configurations from Table 5.2 and Table 6.4 of BAP. To cover those:

- The Supported_Sampling_Frequencies characteristic has the values 0x04 for 16kHz and 0x10 for 24kHz.
- The Supported_Frame_Durations is 0x02 to signify it only supports 10ms frames.
- The Supported_Audio_Channel_Counts values are 0x0001 for the 16kHz QoS options (indices 0 and 1) and 0x0003 for the 24kHz QoS options (indices 3 and 4), indicating that the latter two can support the option of two channels. (Support of two channels is not mandatory, but this is an example of how to use an individual PAC record to expose a specific capability.)
- The Supported_Octets_per_Codec_Frame has values of 0x00280028 and 0x003C003C for 40 and 60 octets, corresponding to the 16_2_n and 24_2_n QoS settings, and finally,
- The Supported_Max_Codec_Frames_Per_SDU indicates that there is only one codec frame per SDU with a value of 0x01. This could have been omitted, as it is the default value. In this Sink PAC Characteristic there is no metadata.

This same information could be provided more compactly in the single PAC record of Table 7.7. The Supported Max Codec Frames Per SDU entry could be omitted as this is the default value.

PAC Record	Record 0 (LTV)
Number of PAC Records	0x01
Codec_ID (LC3)	0x000000000D
Codec_Specific_Capabilities_Length	0x14
Codec_Specific_Capabilities	
Supported Sampling Frequencies	0x 03 01 00 14
Supported Frame Durations	0x 02 02 02
Supported Audio Channel Counts	0x 03 03 00 03
Supported Octets per Frame	0x 05 04 00 28 00 3C
Metadata Length	0x00

Table 7.7 The Sink PAC Record content of Table 7.6 as a single record

These two representations are not completely equivalent. PACS states that an Acceptor must support all possible combinations of a parameter value with all other possible combinations of a parameter value stated in a PAC record. That means that where multiple values are included in a PAC record, as in the example of Table 7.7, then every possible combination is valid, as the Initiator can expand the PAC record into an array of all possible combinations. In theory, that means that an Acceptor exposing the single PAC record of Table 7.7 should, if requested, support a value of 40 octets for 24 kHz sampling and 60 octets for 16 kHz sampling, as well as any octet value between 40 and 60, although these are non-standard. It also means that the ability to support two channels, which we made for Indices 2 and 3 of Table 7.6 would also need to be supported for all configurations.

For the sake of interoperability, there is an assumption that an Initiator should confine itself to using the specific configurations from Tables 5.2 and 6.4 of BAP, which are tried and tested, but an Initiator could choose another combination. That's not good practice, but it is allowed. If not all of those combinations are valid in a device, then the PAC records should be expressed as individual PAC records. However, that may result in the number of records multiplying rapidly, which is not a good thing.

The example above, which is the baseline support needed for BAP, results in four PAC records, which is manageable. Trying to do the same thing for TMAP would require around a hundred individual PAC records, rising to several hundred as GMAP support is added. As we will see shortly (in Section 7.2.1), the unicast Audio Stream Endpoint configuration process can guide the preferred options for configuring an Isochronous Stream, so individual PAC records can be reserved for applications such as vendor specific codec capabilities. More details are provided in Section 3 of PACS.

If an Acceptor supports more than one codec type, at least one Sink or Source PAC characteristic will be required for each, as the Codec_ID field of a PAC record is unique. It is up to the implementation to decide whether all PAC records for a codec are exposed in a single or multiple Sink PAC or Source PAC characteristics. If there are a large number of

Section 7.1 - PACS – the Published Audio Capabilities Service

PAC records, an implementation may want to split them into separate PAC record characteristics to prevent the maximum ATT MTU size being exceeded when reading them.

Sink PAC characteristics do not distinguish between unicast and broadcast streams, so the values apply to both. Source PAC characteristics are ignored for broadcast.

7.1.4 Audio Locations

Both Sink PAC and Source PAC characteristics can have associated Sink_Audio_Locations and Source_Audio_Locations characteristics respectively, although these are optional. The basic concept of Audio Locations was described in Chapter 3. The Sink_Audio_Locations and Source_Audio_Locations characteristics specify which Audio Locations are supported by an Acceptor for received and transmitted audio streams. Each characteristic has a four-octet field which is a bitfield indicating which rendering or capturing locations it supports.

If the characteristic is not present, the device should accept a stream for any Audio Location proposed by an Initiator. In many cases, the values of the characteristics are set by the manufacturer and are read only; a right earbud will only ever be a right earbud, so has a single Sink and/or Source Audio Location of Front Right. Whatever value is set, all Acceptors should accept a mono stream.

There is a subtle difference in the way that the Audio Locations are used in broadcast applications. A Broadcast Transmitter has no knowledge of the capabilities or the Audio Locations of any Acceptors that may be rendering its audio streams. To help guide them to select the correct stream, the Broadcast Transmitter should label each of its streams using the Audio_Channel_Allocation LTV in its BASE. This leaves the choice of which BIS to select to the Broadcast Sink. If a Broadcast Sink wants to receive mono, it should implement a Sink Audio Locations characteristic with the value of 0x00000000. In earlier versions of BAP, this was not allowed. However, that raised a problem where a device's Audio Location was writeable, as it was not possible to signify a change from a defined Audio Location to mono without deleting the characteristic. Now that the value of 0x00000000 is allowed, this characteristic should be used where a broadcast mono application is supported.

This introduces some complexities, especially with speakers. A manufacturer may not want to set speakers to be specifically Left or Right (or any more complex location), but allow a user to select what they want each to be. There are a number of ways of approaching this.

The simplest is to include a physical switch which would set the Audio Location. Alternatively, an app could be provided to perform this, as Audio Locations can optionally be written. In these cases, the speaker would probably only support a single Audio Location value.

A speaker can also set its Sink_Audio_Locations characteristic to both Front Left and Front Right. This requires the speaker to be able to support two Audio Stream Endpoints (see Section 7.2.1). When an Initiator wants to establish a stream, it will decide which Audio Stream

to send to match one of the two Sink Audio Locations. If it finds two speakers with the same values of Sink_Audio_Locations, it would normally send a left stereo stream to the one and a right stereo stream to the other. It would have no indication of which was physically acting as the left or right, leaving the user to work that out. If the Initiator only found one speaker, it could send it both left and right streams, assuming that the speaker could downmix these to mono.

The Source_Audio_Locations characteristic behaves in the same way, although in most cases it is simpler, as it is likely to be limited to a single Audio Location. If the Acceptor supports a single Bluetooth LE Audio Channel, the Initiator would assume that it is a mono microphone. If it supports two Audio Channels, the Initiator would assume it is a stereo microphone.

The use of the Sink and Source Audio Location characteristics, along with Audio Channel Counts provides a lot of flexibility for directing Audio Streams. Much of that is implied, so implementers need to think carefully about the combination of parameters which they use.

Both the Sink and Source Audio Location characteristic values can change, including during a connection. Should that happen during a connection, the audio stream does not need to be terminated. The new values would apply to the next stream establishment process. An application could also decide to change the streams, for instance swapping the left and right channel inputs between the left and right Audio Streams if a user wanted to mirror the stereo effect because of the direction they were facing. This is implementation specific and outside the scope of the specifications.

7.1.5 Supported Audio Contexts

The principle of Audio Contexts was described in Section 3.4. Every Acceptor needs to contain a Supported_Audio_Contexts characteristic, which lists the Context Types which it supports. This is generally a static list, which will only change as a result of a software update which changes the Acceptor's functionality.

The name of Supported Audio Contexts is a little misleading. What this characteristic does is list the Context Types (i.e., the use cases) for which the Acceptor can make itself unavailable, in order to prevent any Initiator trying to establish an Audio Stream for use cases that the Acceptor is not interested in. Support for each Context Type is optional, other than the «Unspecified» Context Type, which has to be supported in every Supported_Audio_Contexts characteristic [BAP 3.5.2.1]. If an Audio Context is not marked as supported, an Initiator can still attempt to establish a stream, by mapping that Context Type to the «Unspecified» Context Type for its Audio Stream. If the Acceptor currently has «Unspecified» set to available (see below), it has no reason to reject that request. As a result of receiving the Audio Stream, an Acceptor may decide to change its Available_Audio_Contexts setting for that Context Type value for future requests.

Section 7.1 - PACS – the Published Audio Capabilities Service

Given this behaviour, an Acceptor should set every Context Type bit in the Supported_Audio_Contexts characteristic to 0b1 if it thinks it will ever have a reason to make that use case unavailable.

7.1.6 Available Bluetooth LE Audio Contexts

An Acceptor uses the Available_Audio_Contexts characteristic to inform an Initiator that it is not available for specific Context Types which it has claimed support for in the Supported_Audio_Contexts characteristic. That may sound a bit contradictory, but this characteristic exists to signal what an Acceptor is currently willing to support. That choice is dynamic. An implementation may decide that it will not accept a phone call while watching TV, or vice versa. When an Initiator wants to start the Audio Stream establishment process, the Context Type of the Audio Stream the Initiator wants to set up must be shown as available in the Available_Audio_Contexts characteristic. Unlike the Supported_Audio_Contexts values, which are static, an Acceptor may update its Available_Audio_Contexts values at any time, notifying connected Initiators when it does so.

The purpose of Available_Audio_Contexts is to help guide transitions between Initiators or applications across multi use case scenarios, by giving them the tools to determine when and whether they should accept a new use case. That's a distinct enhancement to classic audio, where the Audio Source, (typically the phone), made the decisions without input from the peripherals.

To illustrate this, Figure 7.1 shows a typical setting for the Supported_Audio_Contexts and Available_Audio_Contexts characteristics.

		Context Types											
		Emergency Alarm	Alerts	Ringtone	Notifications	Sound effects	Live	Voice assistants	Instructional	Game	Media	Conversational	Unspecified
Bit		11	10	9	8	7	6	5	4	3	2	1	0
Available Audio Contexts		1	0	1	0	0	0	0	1	0	1	1	1
Supported Audio Contexts		1	0	1	0	0	0	0	1	0	1	1	1

Figure 7.1 An example of settings for Supported_Audio_Contexts and Available_Audio_Contexts

In this example, the Acceptor supports «Unspecified», (which is mandatory), along with «Emergency Alarms», «Ringtone», «Conversational», «Instructional» and «Media». It shows all of these as available. If an Initiator wants to start a stream with any of these Context Types, the Acceptor should accept it. The combination of Not Supported and Available is not allowed.

If the Initiator wanted to set up a stream to carry key-press sounds, which are covered by the «Sound Effects» Context Type it can. This is because the Available_Audio_Contexts bitmap shows «Unspecified» is available. As long as «Unspecified» is shown as available in the Acceptor, the Initiator is allowed to map an unavailable Context Type that it wants to use to «Unspecified» in the properties of its Streaming_Audio_Contexts metadata. In which case, the Acceptor must accept the Audio Stream. This is a complex new feature, which may not be supported in the first generation of products, but as Bluetooth LE Audio evolves, we expect it to become increasingly important in providing an enhanced user experience.

		Context Types											
		Emergency Alarm	Alerts	Ringtone	Notifications	Sound effects	Live	Voice assistants	Instructional	Game	Media	Conversational	Unspecified
Bit		11	10	9	8	7	6	5	4	3	2	1	0
Available Audio Contexts		0	0	0	0	0	0	0	1	0	1	1	1
Supported Audio Contexts		1	0	1	0	0	0	0	1	0	1	1	1

Figure 7.2 An example of the Supported_Audio_Contexts and Available_Audio_Contexts with fewer bits set

Figure 7.2 illustrates the value of setting Context Type bits in the Supported_Audio_Contexts characteristic. In this example, «Emergency Alarm» and «Ringtone» are supported, but are not currently marked as available in the Available_Audio_Contexts characteristic. This means that if an Initiator tries to establish a stream associated with these Context Types it will be rejected by the Acceptor as they are specifically set to unavailable. If these bits had not been set as Supported, the Initiator could have mapped them to «Unspecified». However, in this case that is not allowed, as they are set as Supported and not Available. However, the Initiator could still remap streams associated with «Alerts», «Notifications», or any of the other unsupported Audio Contexts to «Unspecified», which the Acceptor should accept.

		Context Types											
		Emergency Alarm	Alerts	Ringtone	Notifications	Sound effects	Live	Voice assistants	Instructional	Game	Media	Conversational	Unspecified
Bit		11	10	9	8	7	6	5	4	3	2	1	0
Available Audio Contexts		0	0	0	0	0	0	0	1	0	1	1	0
Supported Audio Contexts		1	0	1	0	0	0	0	1	0	1	1	1

Figure 7.3 An example of settings with «Unspecified» unavailable

Section 7.1 - PACS – the Published Audio Capabilities Service

Finally, Figure 7.3 shows an example where «Unspecified», is indicated as unavailable (remember that «Unspecified» always has to be supported). Setting «Unspecified» to unavailable prevents the Initiator from mapping any other unavailable Context Type to «Unspecified», so in this case the Acceptor is only available for Audio Streams that are associated with «Instructional», «Media» or «Conversational».

Table 7.8 illustrates this behaviour in terms of how an Initiator interprets the settings. Effectively the Acceptor can set three options:

- a) Allowing an Initiator to proceed with establishing an Audio Stream,
- b) Prohibiting it from starting the Audio Stream, or
- c) Saying it doesn't care – have a go.

The final column maps the use case options from the preceding list.

Supported Audio Contexts	Available Bluetooth LE Audio Contexts	Interpretation for Initiator	Option
0b0	0b0	Audio Stream Context Type may be mapped to «Unspecified»	c)
0b0	0b1	This combination is not allowed	-
0b1	0b0	An Initiator shall not establish an Audio Stream with this Context Type	b)
0b1	0b1	An Initiator may establish an Audio Stream with this Context Type	a)

Table 7.8 Result of Supported and Available Bluetooth LE Audio Context Type settings

Whilst the Supported_Audio_Contexts characteristic is valid for all Clients, an Acceptor may expose different values of its Available_Audio_Contexts characteristic to different Clients. This allows an Acceptor to make a decision on what different Initiators can do, such as controlling which devices can stream media to it or establish an Audio Stream for a phone call on a per Client basis. For example, it could use this to prioritise incoming phone calls where a user has two phones.

How this is managed is up to the implementation, and it is down to the Acceptor to manage the different namespaces required to do this. When implemented, it provides a way for Acceptors to set connection policies based on use cases, which is something that is not possible with Bluetooth Classic Audio profiles. As the Available_Audio_Contexts LTV is likely to change dynamically, it is most likely to be used during the codec configuration state, which we'll come to below.

7.2 ASCS – the Audio Stream Control Service

PACS specifies the way that an Acceptor exposes its properties and what it is available to do at any point in time. For most devices, it will include a wide range of options for codec settings and contexts, whether it can act as a Source and/or Sink, and which Audio Locations it supports. That range needs to be narrowed down to individual sets of parameters to establish Audio Streams, typically because of resource constraints and specific application requirements. This is where the Audio Stream Control Service comes into play, by defining Endpoints for each Audio Stream on each Acceptor. The Audio Stream Control Service is implemented on an Acceptor and defines how an Initiator can configure and establish an Audio Stream between the two of them, using the procedures defined in BAP.

7.2.1 Audio Stream Endpoints

At the heart of the Audio Stream Control Service is the concept of Audio Stream Endpoints. An Audio Stream Endpoint or ASE is defined as the origin or destination of a unicast Audio Stream in an Acceptor. An Initiator does not contain any ASEs – it configures the ASEs on the Acceptors. If audio data flows into an ASE, it's a Sink ASE. If it flows out of it, it's a Source ASE, both of which are described by read-only characteristics in ASCS. An Acceptor must expose at least one ASE for every Audio Channel it is capable of supporting [BAP 3.5.3].

For each ASE, an Acceptor maintains an instance of a state machine for each connected Initiator. The state of an ASE is controlled by each Initiator, although in some cases an Acceptor can autonomously change the state of an ASE. Whereas a Sink and Source PAC provide an Initiator with device wide properties, a Sink and Source ASE represent the current state and properties of each connection. These can be read using the Sink or Source ASE characteristics, which will return the information shown in Table 7.9.

Field	Size (Octets)	Description
ASE_ID	1	A unique, non-zero ID for each client namespace
ASE_State	1	0x00 = Idle 0x01 = Codec Configured 0x02 = QoS configured 0x03 = Enabling 0x04 = Streaming 0x05 = Disabling 0x06 = Releasing
State specific ASE Parameters	varies	Depends on the state (See ASCS Tables 4-2 to 4-5)

Table 7.9 ASE Characteristic format

Section 7.2 - ASCS – the Audio Stream Control Service

Figure 7.4 shows the state machine for a Sink ASE. The state machine for a Source is slightly more complex, as it contains an additional Disabling state. We'll start with the Sink ASE state machine, as the journey through to enabling an ASE is essentially the same up until the Streaming state, then look at the differences when we get to the Disabling state.

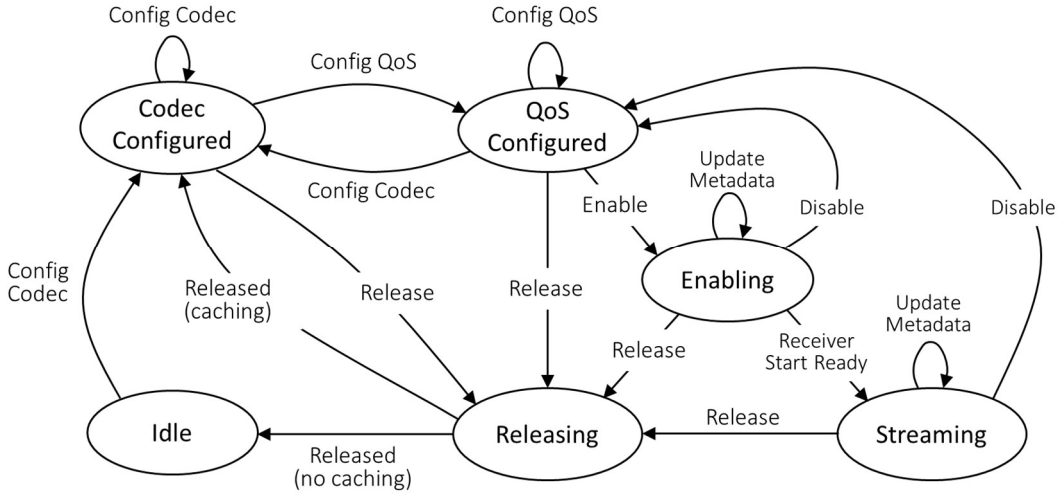


Figure 7.4 The state machine for a Sink ASE

The ASCS defines how an Initiator moves each ASE on an Acceptor through these states, by writing a succession of commands to the ASE Control Point characteristic, using the opcodes listed in Table 7.10. They are defined in Section 4.2 of ASCS.

Opcode	Operation	Description
0x01	Config Codec	Configures an ASE's codec parameters.
0x02	Config QoS	Configures the ASE's preferred QoS parameters.
0x03	Enable	Applies the CIS parameters and starts coupling an ASE to a corresponding CIS.
0x04	Receiver Start Ready	Completes the CIS establishment and signals that an ASE is ready to receive or transmit audio data.
0x05	Disable	Starts to decouple an ASE from its CIS.
0x06	Receiver Stop Ready (Source ASE only)	Signals that the Initiator is ready to stop receiving data and completes decoupling a Source ASE from its CIS.
0x07	Update Metadata	Updates metadata for an ASE.
0x08	Release	Returns an ASE to the Idle or Codec Configured state.

Table 7.10 ASE Control Point operation opcodes

Opcodes 0x01 (Config Codec) and 0x02 (Config QoS) can be used to transition the state of an ASE, or update the configuration of an ASE which is already in that state. Opcode 0x07

updates the metadata, but doesn't change the state. All of the other opcodes result in a transition to another state in the ASE state machine.

An Initiator can perform an ASE Control Point operation on a single ASE, or multiple ASEs on an Acceptor, as long as the ASEs are in the same state. If the CIG includes multiple CISEs, either on one, or multiple Acceptors, the Initiator should ensure that it has collected the relevant information from every ASE on every Acceptor before moving the ASEs on each Acceptor to the Enabling state. That's repeating what CAP says, which is to perform the BAP procedures on all members of a Coordinated Set in the correct order. (In most cases, all of the Acceptors within a CIG will be members of a Coordinated Set. However, ad-hoc sets are allowed in CAP, where the Acceptors can be allocated to work together without being members of a Coordinated Set, in which case the order of configuration is left to the implementation.)

A feature of an ASE is that an Acceptor supports a separate instance for every current connection to an Initiator. If there are multiple Initiators with active ACL connections to an Acceptor, the Acceptor will maintain an independent set of ASE values for each Initiator. It maintains a different ASE_ID namespace for each Initiator. An example of this is a carkit which allows connection to more than one phone. Whilst the ASE_ID must be unique within each namespace, the same ASE_ID can exist in different namespaces. That means that each ASE_ID and its corresponding handle remains unique. Managing that is down to implementation. A change to an ASE in one namespace does not affect the state of the same ASE in a different namespace (although the change may result in an application moving a CIS from one ASE to another one as a result of that change, but that is implementation specific.)

Where Acceptors have built up a connection history to multiple Initiators, they may decide to maintain sets of ASEs with cached configurations. This simplifies the process of transitioning to Audio Streams from a different Initiator. Although the specification allows an Acceptor to have concurrent Audio Streams enabled with multiple Initiators, in practice that requires a lot of resources and complex scheduling, so will probably be rare, at least in early implementations of Bluetooth LE Audio devices. In most cases, utilising cached configurations to allow simple transitions is likely to provide an easier solution.

The corollary of this approach is that Acceptors need to maintain a state machine instance for each connected Initiator. When an Isochronous Stream is disabled, it can remain in the QoS configured state, which can save time when it next needs to establish a stream. The instances of ASEs for that Initiator will reflect this. A consequence of this approach is that another Initiator cannot determine the state of the ASEs for an Isochronous Stream with a different Initiator, as there is no way it can read their ASE instance. All a second Client can do is read the PACS Record(s) or discover the supported profiles to determine the mandatory supported QoS configurations.

Section 7.2 - ASCS – the Audio Stream Control Service

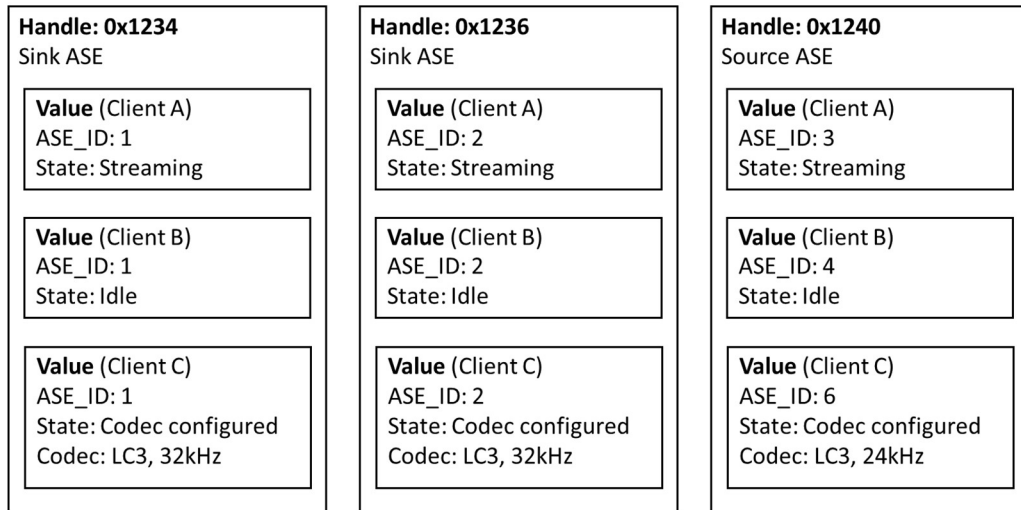


Figure 7.5 An example of exposed ASE states for multiple Clients

Figure 7.5 shows how these principles work, representing a product like a stereo headset, which contains two Sink ASEs and a single Source ASE. It shows how an Acceptor with three ASEs might expose those instantiated states for three different Clients. Currently, only Client A has established streams with the ASEs which have characteristic attribute handles of 0x1234, 0x1236 and 0x1240. If Client A reads the three ASEs, it will learn that all three are in the streaming state.

If Client B were to read the same attribute handles, it would be informed that all three are Idle. Note that the ASE_IDs for Client B may be different, as the Acceptor allocates and maintains a different namespace for each Client.

Finally, when Client C reads them, it will be told that they are in the Codec configured state. This will normally be because Client C has had a previous set of streams enabled, and when they were released, it asked that the Acceptor kept the ASEs in the Codec Configured state to speed up the connection next time Client C wanted to establish a stream. In this state, the codec configuration is exposed, so that Client C can see whether it needs to perform a reconfiguration before moving the ASEs to the QoS configured state. In other states, the Codec Configuration information is not exposed.

An Acceptor must expose at least one ASE for every Audio Channel that it can support. If it supports multiple Bluetooth LE Audio Channels in either direction, it must support at least one ASE for each of those Audio Channels, regardless of whether it uses all of them. If it supports multiplexing, the number of ASEs in each direction must be equal or greater than the number of Audio Location bits set to 0b1 for that direction, divided by the highest number of Audio Channels set in the Audio_Channel_Counts LTV [BAP 3.5.3]. If an Acceptor can support simultaneous Audio Streams from two or more Clients at the same time, it needs to provide an ASE for each Audio Stream. Put more simply, the Acceptor has to support

everything it claims can be thrown at it, with at least one ASE for each Audio Stream.

It might feel as though there is a conflict between the global scope of the PAC record and the more limited configurations of an ASE, but they have different purposes and scope. To understand the relationship between a PAC record and an ASE, it might be helpful to consider an analogy, so we'll resort to food, again. If you think of a restaurant, the Published Audio Capabilities are like a full list of ingredients that are in the kitchen. The ASEs are the individual dishes, which only use a subset of those ingredients. The restaurant's expectation is that people choose the dishes off the menu, because they've normally been developed to suit the use case, which, in the restaurant's case, may be breakfast, lunch, afternoon tea or dinner. However, sometimes the customer might ask for something different. If a diner comes in and asks for a burger with chips, mash, toast, pasta and custard (which I hope is not an item on any menu), BAP takes the view that the customer is always right and allows it. To limit those occasions, it may be helpful to expose individual PAC records which correspond to the use cases supported by the top level profiles. In this restaurant analogy, that would be different menus for different times of day.

Moving around the state machine involves an interplay of BAP issuing a command and ASCS responding. Rather than going into detail about ASCS in isolation, it's more useful to look at the actual process of interactions in setting up a stream.

7.3 BAP – the Basic Audio Profile

ASCS describes the states of an ASE, but not the procedures to use them, as it's a service which resides on a Server. It is a statement of the current state of each Audio Stream Endpoint on an Acceptor. This informs the Initiator as it performs the transitions that move us through the ASE state machine to configure an ASE and enable a CIS. That process is covered in the Basic Audio Profile, which describes the Client (Initiator) behaviour. BAP defines these procedures for both unicast and broadcast and CAP bundles them together. In this chapter we'll confine ourselves to the unicast procedures. In the next chapter, we'll do the same for broadcast.

Setting up a Source ASE is essentially the same process, but with one extra state, which we'll cover at the end.

7.3.1 Moving through the ASE state machine

Moving through the ASE state machine uses a standard process. The Initiator sends a command to the Acceptor's ASE Control Point characteristic, specifying the control operation to be performed (i.e., which state it wants the ASEs to transition to, or which state needs to be updated). The command specifies which ASE or ASEs the command is being applied to and the parameters for that particular state transition.

The command takes the form shown in Table 7.11 and includes a set of operation specific

Section 7.3 - BAP – the Basic Audio Profile

parameters for each state. We'll look at each set of operation specific parameters as we go through the state machine configuration process.

Field	Size (Octets)	Description
Opcode	1	Control Point Operation for the next state or update, (as shown in Table 7.10).
Number of ASEs	1	Total number of ASEs included in this operation
ASE_ID[i]	1	The IDs of those ASEs
Operation specific parameters	Varies	A list of parameters for each of the [i] ASEs

Table 7.11 Basic structure of ASE Control Point command operations for an Initiator

At each stage, the Initiator is made aware of the current capabilities of the Acceptor. As long as it does not request features outside those provided by the Acceptor, the Initiator can expect the Acceptor to honour all of its ASE Control Point command's values. Once it has received each ASE Control Point command, the Acceptor responds with an ASE Control Point characteristic notification, indicating success or otherwise for each of the [i] number of ASE_IDs which were contained in the command, as shown in Table 7.12.

Field	Size (Octets)	Description
Number of ASEs	1	Total number of ASEs included in this operation
ASE_ID[i]	1	The IDs of those ASEs
Response_Code[i]	1	0x00 if successful, otherwise an error response code from Table 5.1 of ASCS
Reason[i]	1	An extended reason code from Table 5.1 of ASCS. 0x00 indicates success. Other codes provide reasons where the operation has failed.

Table 7.12 ASE Control Point characteristic format as notified to its Client

If any of them are rejected or have errors, there is a comprehensive set of error responses to inform the Initiator, so that it can try again. Note that all of these operations are for arrays of [i] ASEs. They can be sent as individual commands for each ASE, but it's more efficient to include all of the ASE_IDs for each Acceptor in one operation.

Assuming there are no issues, the Acceptor then proceeds to update the state of all of its Sink ASE and Source ASE characteristics with the values that the Initiator provided in its ASE Control Point command. If that results in a change to any ASE, the Acceptor will notify the new value for each ASE that has changed.

As the purpose of the ASE Control Point command is to effect a change to an ASE, either by changing its state or updating a parameter value(s), the Initiator will be expecting these

notifications. However, not all ASEs need to be transitioned or updated in each ASE Control Point operation. This means that different ASEs on an Acceptor, associated with the same Initiator, could be in different states. If in doubt, the Initiator should read their ASE Characteristics. (This should not be the case if a CAP Audio Streaming procedure has been used, as that requires all Acceptors to be transitioned to the same state before proceeding with the next command. However, even here, some ASEs may not be in the Enabled or Streaming state if they were configured as “spare” ASEs for future use, which we’ll cover in Section 7.6.)

The Initiator can configure multiple ASEs with its ASE Control Point operation command, but each configured ASE notifies its state independently. For example, if the Initiator were to enable four ASEs for a headset - separate Sink ASEs for left and right stereo and two Source ASEs for a microphone on each side, it would receive one notification of the updated ASE Control Point characteristic and four independent notifications from the two pairs of Sink and Source ASEs. Note that regardless of whether the Sink and Source pairs are implemented as two bidirectional CISEs (which is the most efficient configuration) or four unidirectional CISEs, there are still four ASEs, resulting in four separate notifications.

A simplified sequence chart for this process is shown in Figure 7.6. The same process applies to both Sink ASEs and Source ASEs.

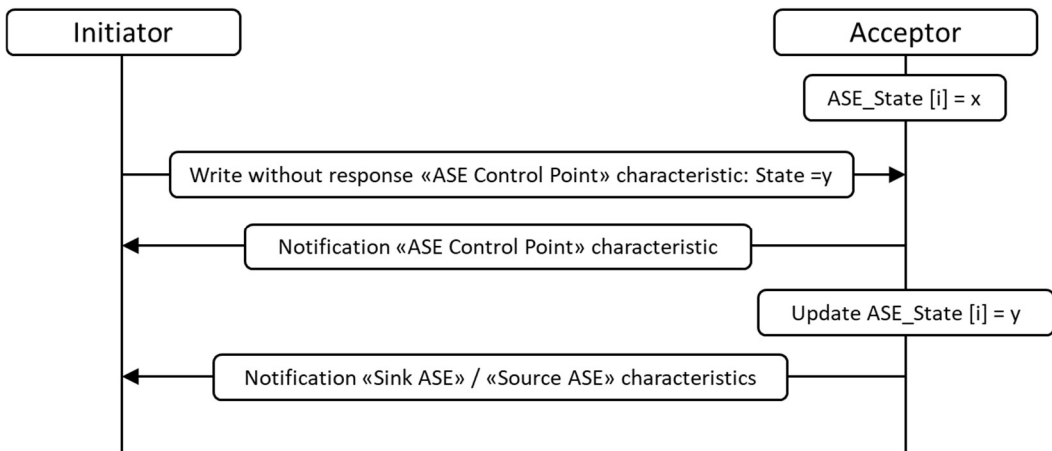


Figure 7.6 Simplified sequence diagram for ASE Control Point operations

The clever (or complicated, depending on your point of view) part of this process is that the parameters in the ASE Control Point command and Sink ASE and Source ASE characteristics are different for each state as you move through the stream establishment process, providing the information that the Initiator needs for its next operation. The different parameters in the first few operations allow the Initiator to understand all of the ASE’s capabilities and gather the information it needs to set up the CIG and its constituent CISEs.

All of the Initiator’s operations have the same basic format, which was shown in Table 7.9 but the State Specific ASE parameters change with each ASE_State. The individual sets of

Section 7.4 - Stepping through the ASE and CIG configuration process

parameters are defined in Tables 4-2 to 4-5 of ASCS for the ASE characteristic notifications and Sections 5.2 – 5.6 for the ASE Control Point commands.

Each step through the ASCS state machine is defined as a specific procedure in BAP. In the next section we'll go through these to see how the process works.

7.4 Stepping through the ASE and CIG configuration process

An Initiator that wants to make a connection, whether in response to a user action, an external event, or a request from an Acceptor, needs to start by determining the capabilities of all of the Acceptors involved in the use case. Assuming that it's the first time the devices have connected, and nothing is cached, there are various options for the Initiator to determine the Acceptor's capabilities, depending on whether:

- the Initiator only requires the basic capabilities mandated by BAP,
- there are mandated features from another top level profile, such as HAP, GMAP or TMAP, in which case it will need to follow CAP as well, or
- it wishes to use optional or proprietary features, which it can configure through BAP.

These options are shown in Table 7.13. In subsequent connections, there may be cached information about the Acceptor's capabilities that allow the Initiator to skip these steps.

Initiator Action	Result
Discover Sink PAC	Acceptor can receive unicast audio with BAP mandatory settings
Discover Sink Audio Locations	Acceptor can receive unicast audio with BAP mandatory settings
Discover Sink ASE	Acceptor can receive unicast audio with BAP mandatory settings
Discover Source PAC	Acceptor can transmit unicast audio with BAP mandatory settings
Discover Source Audio Locations	Acceptor can transmit unicast audio with BAP mandatory settings
Discover Source ASE	Acceptor can transmit unicast audio with BAP mandatory settings
Discover Profile or Service	Acceptor supports additional mandatory requirements
Read Sink PAC	Discover the Acceptor's capabilities for reception
Read Source PAC	Discover the Acceptor's capabilities for transmission

Table 7.13 Options for an Initiator to determine an Acceptor's capabilities

Bluetooth LE Audio allows you to set up everything from first principle by reading the PAC records and then using BAP procedures. If top level profiles are present, it's possible to skip some of the steps, as their presence will tell you more about what an Acceptor will support. For example, if you discover a device exposing TMAP claims support for the Unicast Media Role, you know that it supports 48k sampling, without having to parse its PAC record. This is a short cut to knowing most of the capabilities of the Acceptors. However, the application is free to use other configurations. For example, a broadcast application may decide it needs to save airtime for other purposes, so may elect to configure the audio streams at 24kHz or 32kHz, rather than the 48kHz, which TMAP says must be supported. The mandates in top level profiles only give a guarantee of what a device has to be capable of – it is up to the application to decide what it needs, based on the use case and available resources. .

To illustrate how the process works, let's work through the process of configuring a pair of hearing aids, with one unidirectional CIS and one bidirectional CIS, as illustrated in Figure 7.7. At this stage, we won't make a decision on whether it's for telephony or media; we'll just set up the audio streams.

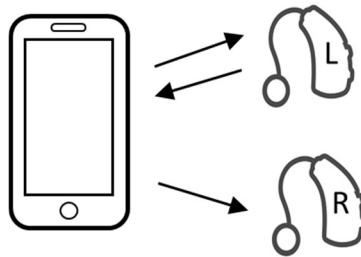


Figure 7.7 *Configuring the ASEs on a pair of hearing aids*

On that basis, let's assume that we're dealing with either TMAP or HAP and want to connect to a left and right pair of Acceptors. Having discovered that HAP or TMAP is supported on at least one of the Acceptors, by discovering their service UUID, the Initiator knows the mandatory codec configurations and QoS settings that are supported. If it's happy to use these, it can jump straight into CAP and run the CAP Connection procedure for non-bonded devices [CAP 8.1.2] to make a connection.

We're making the assumption here that this is the first time the earbuds have been used, so they need to be paired first. The Initiator would determine if the first earbud is a member of a Coordinated Set, and as it is in our example, would then run the CAP procedure preamble [CAP 7.4.1], bonding with both devices in that set.

Having done that, it can start the main business of establishing a connection, using the CAP Unicast Audio Start procedure [CAP 7.3.1.2]. This tells the Initiator to step through the underlying BAP procedures.

First, the Initiator needs to discover all of the Sink and Source ASEs on each Acceptor and

Section 7.4 - Stepping through the ASE and CIG configuration process

confirm that each Acceptor has at least one ASE supporting the correct direction for each Audio Stream it wants to set up. If it's not using the settings mandated in the top level profile, it should also check the PAC characteristics to make sure the Acceptor is capable of supporting the settings it wants to use. These procedures are described in BAP in the Audio role discovery procedure [BAP 5.1], the Audio capability discovery procedure [BAP 5.2] and the ASE_ID discovery procedure [BAP 5.3].

The Audio role discovery and ASE_ID discovery procedures are both mandatory to support, as you can't proceed unless you know there is a suitable ASE to connect to. Audio Contexts discovery [BAP 5.4] is optional, but is necessary if the Initiator wants to use its feature to provide more resolution on how it will handle different use cases. It does that by checking each Acceptor's Supported_Audio_Contexts and Available_Audio_Contexts characteristics, using the Supported_Audio_Contexts procedure [BAP 5.4] and the Available_Audio_Contexts procedure [BAP 5.5]. These were described in Section 7.1.5.

BAP has been designed to provide a fall-back level of interoperability that will allow every Initiator to be able to set up an audio stream with any Acceptor with a reasonable quality of audio, to ensure that they will always be able to connect. This allows an Initiator or an Acceptor to start this connection process off very quickly. Once they have gathered enough information, they move onto configuration of each ASE.

Each ASE has a state machine which needs to be configured, using the single ASE Control Point characteristic, which is used to move around each instance of the ASE State Machine. It is normally good practice (and efficient) to step through all of the ASE configuration states at the same time, i.e. move every ASE to the Codec Configured state, then move them all to the QoS configured state. At the point where they become bound to a CIS, they can still be enabled together, although the CIG state machine allows them to be established separately to allow CISEs to be preconfigured for different applications (see Figure 4.21). We'll just go through the process for a single ASE, but in most cases, there will be multiple ASEs to take through the process. The first step is to configure the codec for each ASE.

7.4.1 The BAP Codec Configuration procedure

Assuming that the requisite Sink and/or Source ASE requirements are met, the Initiator will start to set up the Isochronous Streams by configuring each ASE on each Acceptor. It does this by employing the BAP Codec Configuration Procedure [BAP 5.6.1], where it writes to the ASE Control Point characteristic with the Config_Codec opcode of 0x01, in order to move each ASE to the Codec Configured state. The Initiator has already determined which codec configurations are supported, so it now selects the configuration it wants for its current application and sends that, along with a target latency and PHY. The format of parameters for this operation are defined in Table 5.2 of ASCS and summarized in Table 7.14.

Parameter	Description
Target Latency [i]	0x01 = low latency ¹ 0x02 = balanced latency and reliability 0x03 = high reliability ¹
Target PHY [i]	0x01 = 1M PHY 0x02 = 2M PHY 0x03 = Coded PHY
Codec_ID [i]	Codec configuration (ID, length and configuration), as described in Section 7.1.1 - Sink PAC and Source PAC characteristics.
Codec_Specific_Configuration [i]	
¹ These terms are generic and do not necessarily align with the configurations defined as Low Latency or High Reliability in BAP.	

Table 7.14 State Specific Parameters for the Config Codec operation (Table 5.2 of ASCS)

There are a number of points to highlight here. The first is that these parameters do not need to be the same for each audio stream. With a bidirectional CIS, the parameters, even the PHY and codec, can be different for each direction. In most cases, they are not, but if you're streaming music in one direction and using the return direction for voice control, they may well be. Currently, most profiles require that Bluetooth LE Audio is run using the LE 2M PHY. If the returned parameters differ, the Initiator can accept them, or repeat the command with different values, until it finds some which are acceptable. The Acceptor should issue an Error Code whenever it rejects values, which should help the Initiator understand the problem.

The second point is that the first two of these parameters, which are prefixed with "Target", are recommendations. The Acceptor will use these recommendations to select the QoS parameters it feels best satisfy the requirement and fit its current status, then return those choices in response to this command.

In contrast, the Codec ID and Codec Specific Configuration parameters are a statement of fact and a directive to the ASE to use specific values. The Codec Configuration is generally based on the top level profile being used, which builds on the mandatory options specified in BAP. We looked at mandatory and optional QoS configurations defined by BAP and other profiles in Chapter 5 (Codec and QoS). In most cases, the Initiator will choose one of these predefined configurations, although it could also use settings for an additional codec which is defined in a top layer profile, along with its recommended QoS settings, or a manufacturer specific codec. It could also make up its own configuration, although that runs the risk of poor interoperability. The predefined configurations have been through a lot of testing to ensure they perform well and should always be the preferred option.

Implementers should note that there are two very similar sounding Codec Specific LTV structures. The first, the Codec_Specific_Capabilities LTV structure is used in PAC records and normally indicates a range of capabilities. The second, the Codec_Specific_Configuration

Section 7.4 - Stepping through the ASE and CIG configuration process

LTV structure is used in the Codec Configuration operation (described above) and is for a single specific configuration. The syntax of the individual parameters is subtly different.

Throughout this process there is a degree of the devices dancing around each other, making recommendations, so that the other can provide information on what best suits its current state of resources. This isn't a true negotiation – it's better described as an informed enumeration, allowing the Initiator to populate a command to send to its Controller to configure each CIS. Even then, some of the parameters in the resulting HCI command are only recommendations, with the Controller having the final say over how the CIG is configured.

A third and important point to note is that whilst the Acceptor notifies the range of codec parameters it can support through its PAC records, the Initiator writes specific values, which define how it will configure the encoded Audio Stream. Some of these are defined with different values in the Supported settings the Acceptor sends and the actual settings the Initiator writes. For example, as we saw above, the Supported_Sampling_Frequencies LTV is a bitfield, whereas the Sampling_Frequency LTV is a specific value mapped to a sampling frequency. So, if an ASE supports only 16kHz, the Acceptor will notify a value of 0x04. In response, the Initiator will configure the ASE codec to support 16kHz sampling by writing 0x03. But, to return to the process...

Having obtained the information about the ASEs, their capabilities and availability, the Initiator can issue a single Write without Response command with an opcode of 0x01 which includes an array of all of the ASE_IDs that it wants to configure on an Acceptor, which can cover both directions. The Acceptor will respond by updating and then notifying its ASE Control Point characteristic, including an appropriate response code (0x00 if it's OK). The Acceptor then transitions all of the specified ASEs to the Codec Configured state. Once it has done that, the Acceptor will notify the new state of each ASE, including the following state-specific parameters (summarized in Table 7.15), which are described in Table 4.3 of ASCS.

Field	Description
Framing	Support for framed or unframed ISOAL PDUs
Preferred PHY	A bitfield of values. Normally includes 2Mbps
Preferred Retransmission Number (RTN)	How many times packets should be retransmitted
Maximum Transmit Latency	The maximum allowable delay for Bluetooth transport
Presentation Delay Min & Max	Supported range of Presentation Delay
Preferred Presentation Delay Min & Max	Preferred range of Presentation Delay
Codec Configuration	The codec configuration for this ASE

Table 7.15 Notified parameters following a Config_Codec opcode (Table 4.3 of ASCS)

That concludes the BAP Codec configuration procedure and moves us on to the QoS configuration procedure [BAP 5.6.2].

7.4.2 The BAP QoS configuration procedure

At the end of the Codec configuration procedure, the Initiator's Host will compare the configuration parameters that each Acceptor notified at the end of the Codec configuration procedure, with the requirements of its current application, and decide what values it wants to use within the limits of what it has been told.

Before issuing the Config QoS command, it is good practice for the Initiator to send these parameters to its Controller using the LE Set CIG Parameters HCI command. This will confirm that the selected parameter set can be supported. Remember that these are use case related recommendations which inform the scheduling algorithms in the Controller. The actual values the Controller chooses may differ slightly. The parameters used in this HCI command are shown in Table 7.16. Full details of the command are in the Core in Vol 4, Part E, Section 7.8.98.

Parameter	Description
CIG_ID	Assigned by the Initiator's Host
SDU_Interval_C_To_P	Time interval between SDUs generated from the Initiator's Host
SDU_Interval_P_To_C	Time interval between SDUs generated from the Acceptor's Host
Worst_Case_SCA	Worst case Sleep Clock Accuracy of all of the Acceptors in the CIG.
<i>Packing</i>	<i>Preferred packing (sequential vs interleaved)</i>
Framing	Framed if set to 1, <i>otherwise up to the Controller on a per-CIS basis.</i>
Max_Transport_Latency_C_To_P	The maximum transport latency for each direction between Central and Peripheral or vice versa.
Max_Transport_Latency_P_To_C	
CIS_Count	The number of CISes in this instance of this command.
CIS_ID[i]	Unique ID for each CIS in this CIG.
Max_SDU_C_To_P[i]	The maximum sizes of SDUs from the Initiator's and Acceptor's Hosts.
Max_SDU_P_To_C[i]	
<i>PHY_C_To_P[i]</i>	<i>Bitfield indicating the possible PHYs to use for each direction. If more than one bit is set, the Controller decides.</i>
<i>PHY_P_To_C[i]</i>	
<i>RTN_C_To_P[i]</i>	<i>The Preferred number of retransmissions in each direction.</i>
<i>RTN_P_To_C[i]</i>	<i>The Controller can ignore this.</i>

Table 7.16 LE Set CIG Parameters HCI parameters

Section 7.4 - Stepping through the ASE and CIG configuration process

In Table 7.16, the parameters highlighted in *italics* are recommendations to the Controller, which it may choose to ignore.

As we saw in Chapter 4, an application cannot force specific values for the Link Layer. Reiterating what is happening here, the Initiator and Acceptors have exchanged information which allowed the Initiator to determine what to put into its HCI command⁵⁷ to instruct the Core to schedule CISes that best meet the application requirements.

There are two values which are required in the LE Set CIG Parameters HCI command (Table 7.16) which don't come from the ASE configuration process. The first is the *Worst_Case_SCA*, which is the worst case sleep clock accuracy of all of the Acceptors. The Initiator needs to determine this by requesting the current value of each device's sleep clock accuracy before issuing the command, normally by using the *LE_Request_Peer_SCA* HCI command.

The second is the *Packing* parameter, which determines whether the CIS events will be sent sequentially or be interleaved. If set to 0, they will be sequential; if set to 1 they will be interleaved. An application may have a preference, but ultimately, the decision is left to the Controller.

Normally the *Framing* parameter value is set to 0, so that the Controller can decide, but some profiles such as HAP, may mandate that in some cases it be set to 1, to force them to be framed. The recommended QoS settings for unicast Audio Streams in Table 5.2 of BAP state that they should all be unframed apart from those with a 44.1kHz sampling frequency, which should be framed.

At this stage the Controller does not inform either the Initiator or Acceptor's Host of the values it has chosen – that information will be conveyed when the CISes are established. Instead, the Initiator's Host only gets confirmation that its requested configuration can be scheduled, along with Connection Handles for each of the CISes.

Once the *HCI_Command_Complete* event has been received, confirming that the CISes can be scheduled, the Initiator should send the ASE Control Point operation command with the opcode set to 0x02 to each of its Acceptors in turn, with the operation specific parameters defined in ASCS Table 5.3 and shown below in Table 7.17. This will move the ASEs to the QoS configured state. The values that the Initiator uses in the *Config QoS* command replicate the parameter values that it used in its *HCI LE_Set_CIG_Parameters* command.

⁵⁷ Bluetooth implementations use the HCI commands for testing to check that they are compliant, but they don't need to be exposed in production products. Silicon and stack vendors may provide alternative APIs to provide this functionality, but understanding the HCI commands helps to demonstrate how the process works.

Parameter	Value
Number of ASEs	i (must be at least 1)
CIG_ID	Values used in the HCI LE_Set_CIG_Parameters command. (Note: At this stage, these may not be the values which the Controller will schedule and use.)
CIS_ID [i]	
SDU_Interval [i]	
Framing [i]	
PHY [i]	
Max_SDU [i]	
Retransmission_Number [i]	
Max_Transport_Latency [i]	
Presentation_Delay [i]	

Table 7.17 CIS and Presentation Delay values used with 0x02 Config_QoS opcode [ASCS Table 5.3]

The Presentation Delay is the only parameter which is not included in the HCI LE Set CIG Parameters command. Instead, it is sent directly to the Acceptor in this Config_QoS operation, as it is an application specific value that is not related to the CIG parameters, apart from the fact that it has to allow enough time for the respective audio decoding (for an Audio Sink) or audio encoding (for an audio source). In almost all applications, the value of Presentation Delay will be identical for all of the Sink ASEs. The reason the Presentation Delay value is the same for each Sink ASE is that this determines the point at which audio will be rendered. For earbuds and hearing aids, that would always be at the same time. In some situations, such as where there are multiple speakers in a conference room or auditorium, there might be a requirement to assign different values to cope with audio latency related to their relative position in the room, but that is not the norm. Similarly, the Presentation Delay for multiple Source ASEs should be the same for all Source ASEs, although this value will normally be different to the value set for the Sink ASEs.

Source ASEs are likely to use a different value of Presentation Delay to the value for the Sink ASEs, not least because the overall LC3 encode time, which is part of the Presentation Delay, is significantly greater than the decode time (around 13ms, as opposed to 2ms). So, more time is required to encompass that encoding. However, all Source ASEs would normally use the same value as each other to ensure that they capture audio at the same point in time. If the Initiator knows this has happened, it makes it far easier to combine the inputs from multiple microphones.

The Presentation Delay values chosen by the Initiator for rendering should be within the preferred Presentation Delay ranges exposed by the Sink ASEs, and the capturing values within those exposed by the Source ASEs. The values should not extend beyond the common range within the Max and Min values that were exposed for each direction by the set of Acceptors and should ideally be within the range of preferred Max and Min values (see Table 7.15). If the Context Type is «Live» or «Conversational» it should use the lowest common minimum value.

Section 7.4 - Stepping through the ASE and CIG configuration process

As before, the Initiator sends these in a Write without Response command, after which it receives a notification of the updated ASE Control Point characteristic, confirming that each ASE has moved to the QoS Configured state, followed by notifications of the updated ASE characteristic for each ASE. The parameters in these notifications reflect the values set in the previous ASE Control Point operation:

Parameter	Value
CIG_ID	Values set by the Initiator in its ASE Control Point operation. (Note: At this stage, these may not be the values which the Controller has actually scheduled.)
CIS_ID	
SDU_Interval	
Framing	
PHY	
Max_SDU	
Retransmission_Number	
Max_Transport_Latency	
Presentation_Delay	

Table 7.18 Parameters returned by an Acceptor after receiving a Config QoS Opcode (0x02)

Repeating the obvious once again, the Initiator should step through each state for all of the ASEs it intends to use on all of the Acceptors before proceeding to the next state, i.e., it takes them all to the Codec configured state, then all to the QoS configured state, etc. That's purely common sense, because the Initiator needs to obtain information from all of them to ensure that it sets configuration parameters for the ASEs that work across all of the Acceptors. However, just to be sure, CAP mandates it.

If multiple Acceptors or ASEs don't provide a consistent set of parameters, the Initiator can repeat each step of the Config and QoS configuration process for one or more of them, including going back from the QoS configured state to the Codec Configured state. However, the expectation is that this procedure would normally be a single step process, which is performed once only for each set of ASEs, as it's more efficient if it keeps all of the state changes in sync. So, the process steps are:

- The Initiator sends a command with target values and explicit values for multiple ASEs
- The Acceptor confirms the new state for all of those ASEs (which may include a failure code) by notifying its ASE Control Point Characteristic
- The Acceptor separately notifies its preferred values for each ASE using ASE characteristics. (This is the point where it states what has been configured and its preferences for the next configuration step.)

At any point, the Initiator can check an individual ASE characteristic. The parameters will indicate the state of the ASE, along with values that are relevant to that state. If an Initiator

reads an ASE_ID in the Idle state, the only information it will receive is the ASE_ID and an ASE_State value of 0, indicating the Idle state. The most common reason for reading an ASE characteristic is to confirm a previous ASE Control Point operation when an expected notification has not been received.

When the Initiator's Host sends the HCI LE Set CIG Parameters command to its Controller, it results in the CIG moving to its Configured state. Figure 7.8 reminds us of that. Remember also that the CIG state machine resides on the Initiator, while the ASE state machines are located on the Acceptors.

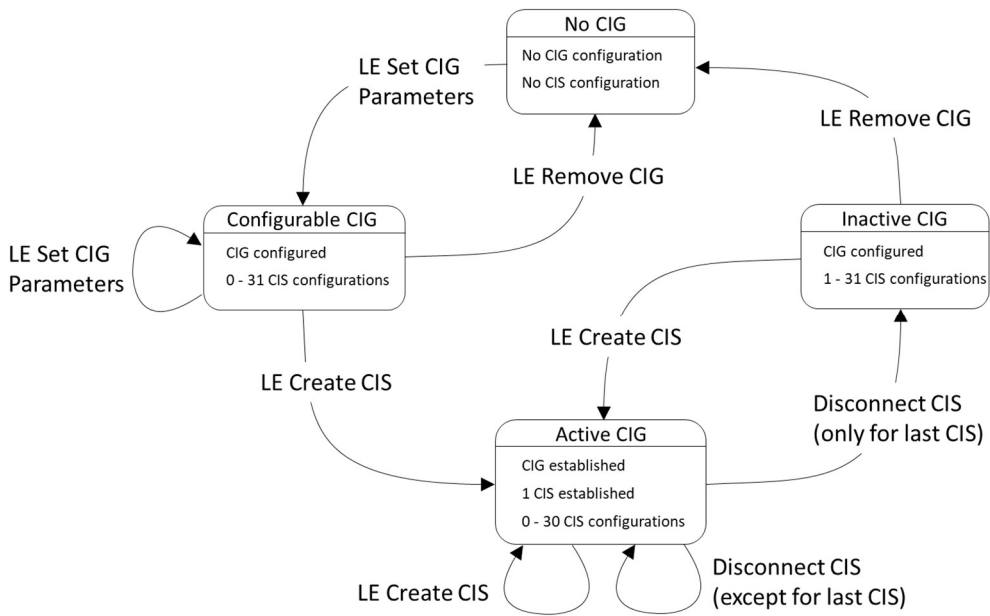


Figure 7.8 The CIG State Machine

The Initiator's Host can still modify the properties of a CIS or add further CISes to the CIG at this point, by resending an HCI LE Set CIG Parameters command for specific CISes, using its array structure. If it does, it will need to update the relevant ASEs using the Config QoS command by writing to the ASE using the array capability of the ASE Control Point characteristic.

Once all of the ASEs for all of the Acceptors are in the QoS configured state and the CIG is configured, we can start to enable the Audio Streams.

7.4.3 Enabling an ASE and a CIG

Having configured all of the ASEs, the Initiator will have all of the information it needs to instruct its Controller to enable the CIG and constituent CISes. The Acceptor's Host knows the main parameters of the Isochronous Channels, although some values are still provisional, as the actual RTN value will depend on the scheduling. If the Initiator's Controller is also

Section 7.4 - Stepping through the ASE and CIG configuration process

having to support other wireless connections, it may have to compromise to allocate airtime between the different transceivers and set a lower number of retransmissions than its Host requested. At the point that each CIS is established, the final settings will be notified to the respective Hosts of the Initiator and each Acceptor.

The Initiator can now invoke the Enabling an ASE procedure defined in BAP 5.6, by writing to each Acceptor's ASE Control Point characteristic with the opcode set to 0x03 (Enable), using the parameters shown in Table 7.19. This moves the CIG to its Active state. From this point, the Initiator cannot change the CIG, CIS or ASE configurations, with the exception of the Audio Stream metadata.

Parameter	Description
Number of ASEs	Total number of ASEs to be enabled (i)
ASE_ID [i]	The specific ASEs which are being enabled
Metadata Length [i]	Length of metadata for ASE [i]
Metadata [i]	LTV formatted metadata. This is most commonly the Streaming_Audio_Contexts and CCID_List LTV structures, but can include other LTV structures.

Table 7.19 State Specific Parameters for the Enabling operation – Opcode 0x03 (Table 5.4 of ASCS)

CAP mandates that the metadata in the Enabling operation includes the Streaming_Audio_Contexts, with the correct values set for each unicast Audio Stream. If there is a content control procedure associated with the Audio Stream, then the Metadata in the Enabling command must also include the CCID_List LTV structure. [CAP 7.3.1.2.6]

Each Acceptor will notify its ASE Control Point characteristic, move its ASEs to the Enabling state, then, in turn, notify the ASE characteristic for each of the ASEs specified in the ASE Control Point command, returning their CIG and CIS IDs, along with any metadata written by the Initiator, as shown in Table 7.20.

Parameter	Value
CIG_ID	Values set by the Initiator in its previous Config QoS operation.
CIS_ID	
Metadata Length	(0 if there is no metadata for this ASE)
Metadata	LTV structures for the Sink or Source ASE

Table 7.20 Additional parameters for the ASE characteristic when in the Enabling, Streaming and Disabling states

The metadata values for an ASE can only be set or updated by the Initiator. An Acceptor cannot make changes to these, even if it is acting as the Audio Source. An Acceptor can update its Available_Audio_Contexts value at any point, but these do not appear in the ASE metadata. Nor do they result in the termination of a current stream. Any change in the Available_Audio_Contexts is only used for subsequent connection attempts.

Now that the ASEs are in their Enabling state, it is time to enable the required CISes and couple them to the ASEs. Remember from Chapter 4, that not every configured CIS included in the CIG needs to be established, as an Initiator can configure CISes which it can swap in and out as its demands change. The Initiator enables the CISes it requires by invoking the Connected Isochronous Stream Central Establishment procedure from the Core [Vol 3, Part C, Section 9.3.13]. The Link Layer will generate a CIS_Request to the Acceptor, which establishes the CIS using the Isochronous Stream Peripheral Establishment procedure from the Core [Vol 3, Part C, Section 9.3.14]. At this point, the Initiator will start transmitting packets with zero length PDUs. If it is a bidirectional CIS, both Sink and Source ASEs need to be established.

There is a condition on the behaviour of the ASE at this point which implementors need to be aware of. If the CIS existed before the enabling operation, the transition from the Enabling state to the Streaming state is not notified. Instead, the Acceptor autonomously transitions the ASE to its Streaming state, without the need for the Initiator to write the Streaming command. This is most likely to occur if the Acceptor is reusing an existing configuration and the original CIG had never been disabled.

As we saw in the CIG state machine in Chapter 4, once the CIG has been enabled, the CIS configurations cannot be changed, so any change to the codec configuration, QoS parameters or Presentation Delay would require the CIG to be terminated, and the configuration process restarted. An individual CIS can be disconnected or restored (using the LE Create CIS command), but only their metadata can be updated once they are in the Enabling or Streaming state.

7.4.4 Audio data paths

The Isochronous Streams are now up and running, but we haven't connected any audio data – the Isochronous Channels are just conveying empty packets. The next step, if it has not already been done, is to connect a data path for each ASE, which uses the Audio Data Path Setup Procedure (BAP 5.6.3.1), which in turn uses the LE Setup ISO Data Path HCI command.

Bluetooth LE Audio provides a lot of flexibility for the audio data path and where the codec is located. The codec can reside in the Host or in the Controller, and the audio data can come through HCI, or more typically via PCM or a vendor proprietary route.

Section 7.4 - Stepping through the ASE and CIG configuration process

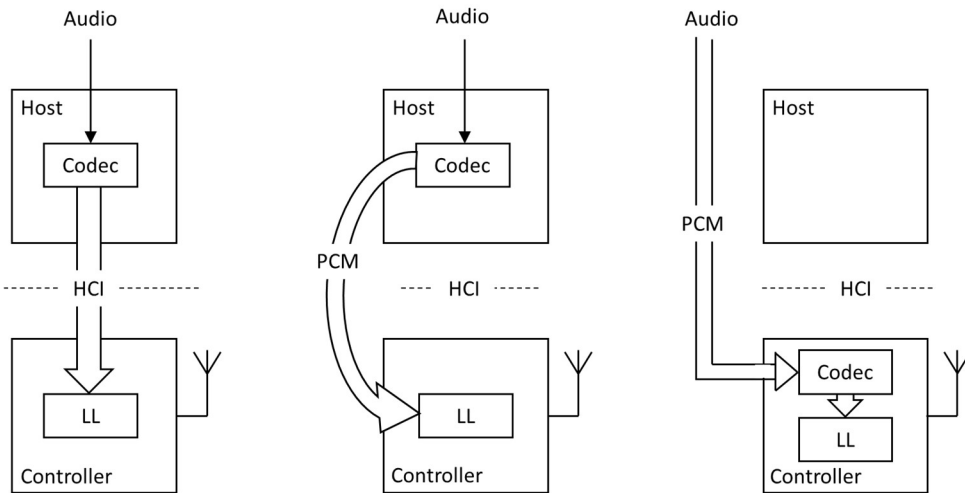


Figure 7.9 Common Audio Data Path configurations showing the transport of codec frames

Figure 7.9 shows three of the most common data path configurations, showing that the codec can be implemented in the Host or the Controller. The encoded codec frames are typically routed from the codec via a PCM interface, but, if encoded in the Host, they can also be transported over HCI. To set up each data path, the Initiator and Acceptors will both use the LE Setup ISO Data Path HCI command to bind the codec configurations that were written during the Config Codec state to the Connection Handle of each CIS, specifying the direction depending on whether it is a Source or Sink ASE. The data path configuration may be different in the Initiator and Acceptor, as the codec location and data path implementation will normally depend on the specific chipset being used. As the data path setup is internal to each device, that's an implementation choice. This level of detail is normally hidden from the application developer below a more general API, but it's useful to know what happens at each stage of the configuration process.

Up to this point, the process is common for both Sink and Source ASEs. Now the two processes diverge. They're still on the same path within the State Machines, but there is a difference in the order of commands.

For a Sink ASE, once the Acceptor has successfully set up its data path, it may autonomously move the ASE to the Streaming state, then notify each Sink ASE characteristic to the Initiator with the ASE state code set to 0x05 (Streaming). As soon as it receives this notification point the Initiator can start streaming audio packets to that ASE Sink.

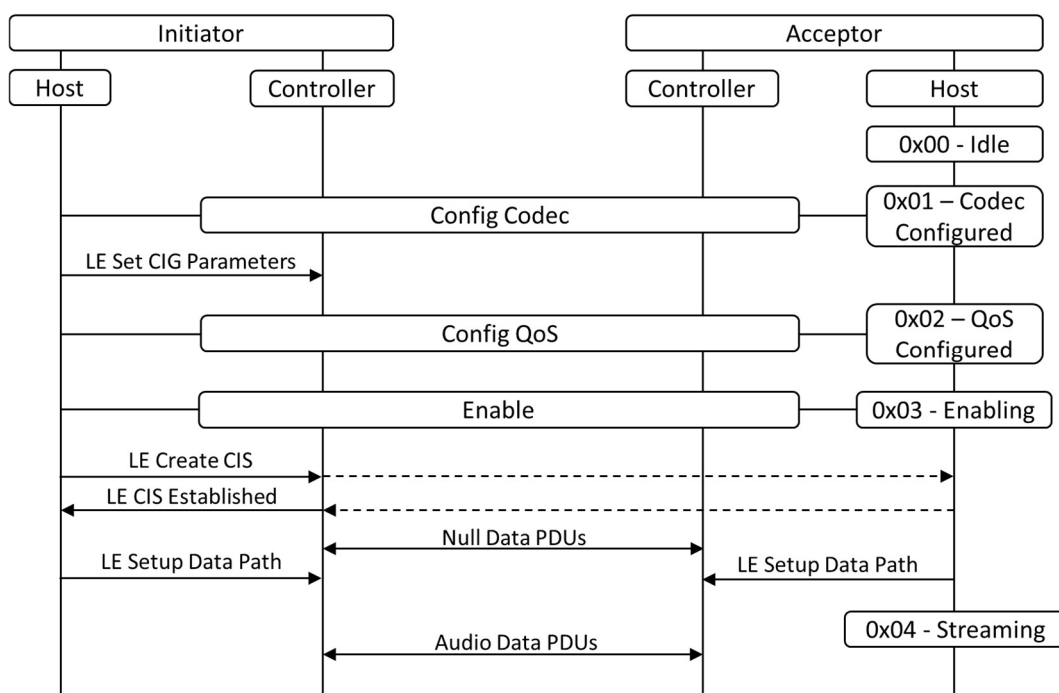


Figure 7.10 Establishment of a Sink ASE, showing the ASE states

Figure 7.10 shows a very simplified message sequence chart for establishing a Sink ASE, showing the overall sequence and the Acceptor's autonomous transition to the Streaming state. More detailed MSCs are provided in BAP Section 5.6.3.

For Source ASEs, the Acceptor needs to wait for the Initiator to confirm that it is ready to receive data from that ASE. Once it has received the ASE characteristic notification from the Acceptor, and as soon as it is ready to transmit audio data, the Initiator will write the ASE Control Point characteristic for that ASE with the opcode set to 0x04 (Receive Start Ready). (Note that Sink ASEs do not need to be included in the array of ASEs in this command, as they will independently move to the Streaming state.) At this point, the Acceptor will transition the Source ASE to the Streaming state, notify its ASE characteristic and commence audio streaming.

A corresponding MSC for a Source ASE is shown in Figure 7.11. Here the Initiator needs to issue the Receiver Start Ready command to the Acceptor's ASE Control Point characteristic to initiate the start of audio data packet transfer.

Section 7.4 - Stepping through the ASE and CIG configuration process

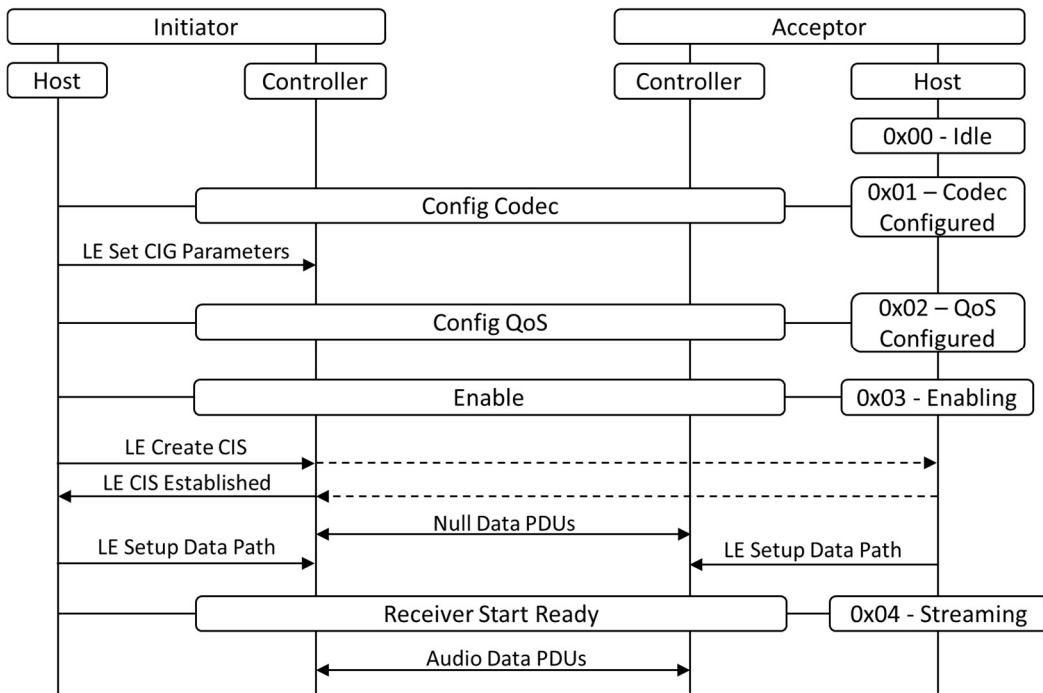


Figure 7.11 Establishment of a Source ASE, showing the ASE states

7.4.5 Updating unicast metadata

Whilst in the Enabling or Streaming states, an Initiator can update the metadata for any ASE by writing to the ASE Control Point with an opcode of 0x07 (Update Metadata) [BAP 5.6.4]. This is a convenient way of reusing an ASE for a different audio application, without having to tear down and re-establish any Audio Streams.

This is described in the CAP Unicast Audio Update procedure [CAP 7.3.1.3] and allows an ASE to be reused, keeping its current configuration, but changing the Context Type and CCID_List metadata. A typical application would be to use the same stream to move from a phone call to a music player on the same phone. There are inherent limitations, which is that the codec configuration cannot be changed. Another issue is that a phone call is normally bidirectional, whereas music streams are unidirectional. However, during the configuration process, enough ASEs could be configured to cope with this. At the point of transition of the use case, the Initiator could disable and enable ASEs until it has the number of configured ASEs which it needs, updating their metadata in the process. Note that in the case of a bidirectional CIS, an ASE might be disabled, but the CIS would remain enabled to support the other direction and its ASE. It is a good example of how the Bluetooth Classic Audio multi-profile issues have been designed out by adding flexibility for Audio Streams to be repurposed and reused.

7.4.6 Ending a unicast stream

The process of ending a unicast stream is where the Sink and Source ASE state machines diverge. We'll cover the Source ASE in the next section. The difference is that a Source ASE has a Disabling state, whereas the Sink ASE does not.

To stop streaming to a Sink ASE, the ASE needs to transition back to the QoS Configured state, using the CAP Unicast Audio Stop procedure [CAP 7.3.1.4]. This calls the BAP Disabling an ASE procedure [BAP 5.6.5], where the Initiator writes to an ASE with the 0x05 (Disable) command. At this point, the Initiator stops transmitting audio data to the ASE. If the CIS is bidirectional and the Source ASE has not been disabled, the Initiator will continue to transmit null PDUs to allow the Acceptor to return its audio data.

The associated CIS is not automatically disabled at this point. If the Initiator wishes to remove it, it should use the HCI_Disconnect command [Core Vol 4, Part E, 7.1.6]. As disabling the Sink ASE only moves it to the QoS configured state, the ASE can be re-established at a later point by writing the Enable opcode to the ASE Control Point characteristic. If the CIS has been disabled using the HCI Disconnect it can still be restored using the HCI LE Create CIS command. However, the CIG remains in the Active state.

If there is no intention of reusing the ASE, it can be moved to the Releasing state by performing the BAP Releasing an ASE procedure [BAP 5.6.6]. Once the ASE is in the Releasing state, the Acceptor autonomously transitions it to the Idle state, or, if it wants to simplify the next ASE configuration cycle, it can cache the codec configuration by returning it to the Codec configured state.

Once all of the ASE's associated with a CIS have been disabled, the Initiator can disable any CISes which are still enabled and tear down the associated data paths. When all of the CISes in a CIG (across all Acceptors) have been disabled, the CIG will transition to the Inactive state and can then be removed.

Section 7.4 - Stepping through the ASE and CIG configuration process

7.4.7 The Source ASE state machine

A Source ASE behaves slightly differently, as we need a confirmation from the Initiator to ensure a clean transition. This introduces a Disabling state. It is only used for Source ASEs, and is shown in the Source ASE state machine of Figure 7.12

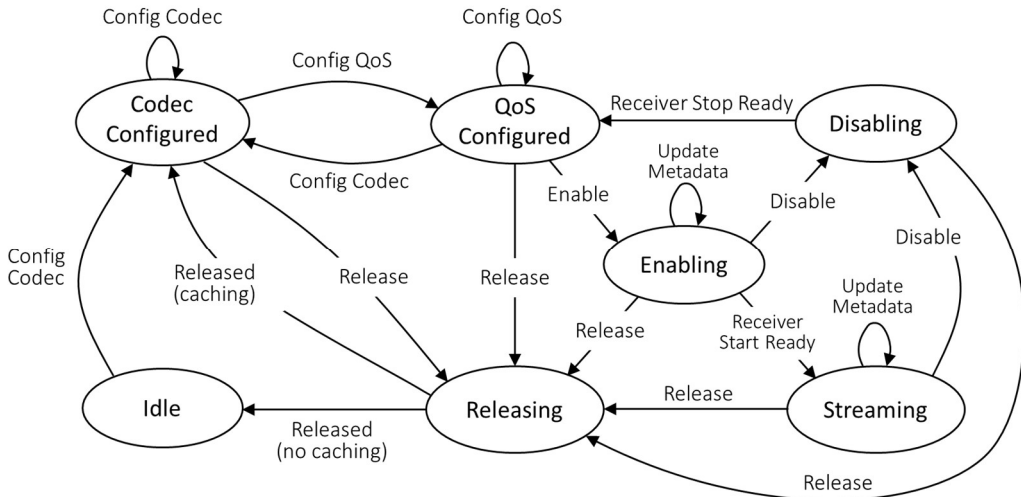


Figure 7.12 State machine for a Source ASE

For a Source ASE, the Acceptor needs confirmation from the Initiator that the Initiator is ready to stop receiving audio data. The Acceptor can autonomously stop streaming, but the Receiver Stop Ready command results in a clean termination of streaming. This is important if there is a potential requirement to reuse the ASE. Having notified the fact that it is in the Disabling state, the Acceptor should wait for a command from the Initiator to tell it to stop streaming and move to the QoS configured state. From there, the Source ASE can be moved to the Releasing State, but if it does, it will not be possible to reuse the CIS without taking the ASE through the state machine again.

Once in the Releasing state, whether by a direct transition from the Streaming state for a Sink ASE, a direct transition from the Disabling state for a Source ASE, or being released from the QoS configured state for either, both Initiator and Acceptor should tear down their data paths and terminate the CIS. The Acceptor can choose to transition to either the Idle or Codec Configured states, both of which operations are performed autonomously. If the CIS is bidirectional, it should not be terminated until both Sink and Source ASEs are in the Releasing State. When the final CIS is terminated, the CIG moves from the Inactive CIG state to the No CIG state.

7.4.8 A summary of the CIS configuration process

To summarise the process described above, Figure 7.13 illustrates the sequence of step in configuring a CIG with a single Acceptor.

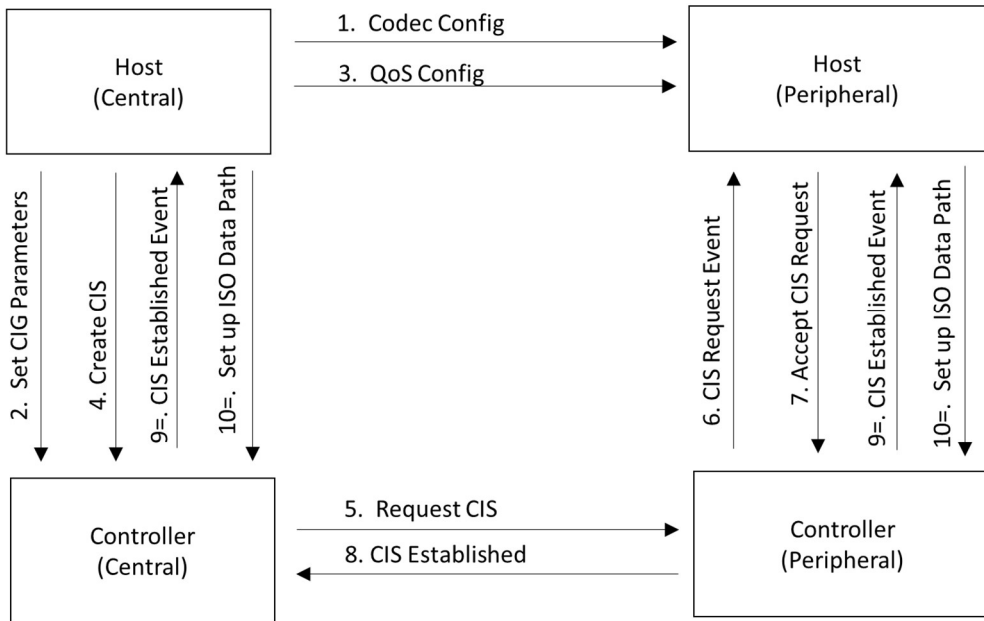


Figure 7.13 The overall process of establishing a CIG

Step 1. The Initiator's Host sends the Codec Config opcode to its Acceptor. If there are multiple ASEs required for the application, it should do this for each ASE. If there are multiple Acceptors, as would be the case with earbuds, hearing aids and some speakers, it should complete this step for each Acceptor.

Step 2. The Initiator collates all of the responses it has received from its Acceptors and issues a HCI Set CIG parameter command to its Controller. If the Controller responds saying that it can accommodate the request, the Initiator proceeds to Step 3. Otherwise, it may need to repeat Step 1 with a relaxed set of requirements.

Step 3. The Initiator's Host sends the QoS Config opcode to its Acceptor(s) and collects responses.

Step 4. The Initiator uses this information to issue an HCI Create CIS command, which should result in the Controller starting the process to establish one of more of the CISes in the CIG.

Step 5. The Initiator's Controller configures itself to support the CIG and requests each Acceptor to create accept the relevant CISes.

Section 7.4 - Stepping through the ASE and CIG configuration process

Step 6. Each Acceptor's Controller asks its host to accept the CIS. The parameters may be different from those it had anticipated, as the Initiator may have made changes to the configuration in Step 5.

Step 7. If it is able to accept the requested CISes, each Acceptors' Host informs its Controller that it should confirm this to the Initiator,

Step 8. Each Acceptor Controller informs the Initiator's Controller that it has established the CISes.

Step 9. Each device's Controller informs its Host that the CISes have been established.

Step 10. Each device's Host sets up the ISO appropriate data paths for its audio data and informs the audio application that it can proceed to start streaming data.

7.4.9 Autonomous operations on an ASE

We've already seen that an Acceptor can autonomously transition a Sink ASE from the Enabling State to the Streaming state and from the Releasing state to the Idle state or the Codec Configured state. These are not the only occasions where an Acceptor can act autonomously. With the exception of the transitions shown in Table 7.21, all of the state machine transitions can be performed by either the Acceptor or Initiator. However, in most cases, moving around the state machine is performed as described above.

ASE Type	Current State	Next State	Initiating Device
Sink and Source	Codec Configured	QoS Configured	Initiator
Sink and Source	QoS Configured	QoS Configured	Initiator
Sink and Source	QoS Configured	Enabling	Initiator
Source	Disabling	QoS Configured	Initiator
Sink and Source	Releasing	Codec Configured	Acceptor
Sink and Source	Releasing	Idle	Acceptor

Table 7.21 ASE transitions which are confined to Initiator or Acceptor actions

7.4.10 ACL link loss

If the ACL link is lost between an Acceptor and an Initiator, all CISes to that Acceptor are disconnected. Where there are other Acceptors involved in the CIG, the Acceptor should move all of the ASEs associated with the lost ACL link to the QoS configured state, so that the CISes can be reenabled if the link returns. The Acceptor will notify this change of state, although it is questionable whether the Initiator will receive the notification.

As Acceptors are not necessarily aware of the existence of any other Acceptor(s), they may not know whether the CIG is still active or streaming to other Acceptors. If they don't receive a reconnection request after a link loss, they may employ an implementation specific timeout

to return their ASEs to the Idle state. On a later reconnection of the ACL link, the Initiator should read the ASE characteristics of that Acceptor to check its state. If it was released from the QoS configured state because of a timeout, the Initiator will normally need to tear down and re-establish the entire CIG. Remember that there is an asymmetry between Initiator and Acceptor – the ASE state machine resides on the Acceptor, whereas the CIG state machine is on the Initiator. An Acceptor is never aware of the CIG state; it is what the Initiator, which has a global view of the ACL connection status, uses to drive the ASE states.

7.5 Handling missing Acceptors

Before starting to configure ASEs, CAP requires that an Initiator connects to all of the available Acceptors in the target Coordinated Set. In the real world, there will be occasions when not all of them are present, either because one of them is turned off, missing, or out of range. In this case, the Initiator should go ahead with setting up the members of the Coordinated Set which it can find. How long it waits before this happens and whether it involves user feedback, as well as the choice of Audio Channels to send to the available Acceptor are all implementation specific. The Initiator should continue to search for the missing Acceptors, adding them in when it discovers them.

The nature of a CIG is that once it is Active, additional CISEs cannot be configured. Hence, if the Initiator only configures CISEs for the Acceptors it can find, then, if missing ones turn up, and no CISE had been scheduled for them, it would have to tear down the CIG, reconfigure it and re-establish the CISEs, which will disrupt the Audio Streams.

To prevent that break in the audio, an Initiator can schedule the CIG with cached values for any missing ASEs. If it detects the presence of the missing Acceptor at a later point, it can configure their ASEs, then enable the associated CISEs in the active CIG, (as they have already been scheduled), and start the transfer of audio data. CAP doesn't describe this process; it's an extension of functionality by using BAP procedures in addition to CAP ones, but it can result in a better user experience.

7.6 Preconfiguring CISEs

A similar case exists where an Initiator knows that an Acceptor is likely to participate in a number of different use cases which have different ASE configurations. A common example of this is streaming music (the analogue of A2DP), which just needs Sink ASEs to enable the CISEs carrying audio from Initiator to Acceptor, but which may be interrupted by incoming phone calls, where a return audio stream from the microphone(s) is required. To make this transition faster, the Initiator can configure a set of ASEs which fulfil both use cases, i.e., two Sink ASEs and two Source ASE, so that when it wants to transition between the two applications, all it needs to do is enable or disable the Source ASEs, as opposed to tearing everything down and restarting.

The downside to this is that the airtime for the return Isochronous Stream is always allocated,

Section 7.7 - Who's in charge?

although it may never be used. A stereo 48_2_2 stream for High Reliability takes up around 63% of the airtime. If it is scheduled to include a 32_2_2 return stream for each CIS, that increases to 90%. By comparison, a bidirectional 32_2_2 stereo stream takes only 41% of airtime. There is also a discrepancy in the latency requirements of the two applications. For music streaming, which generally uses the High Reliability QoS settings, the overall latency for a 48_2_2 stream, with a 32_2_2 return is just over 140ms. That's a lot longer than the 56ms you would get with a bidirectional, Low Latency 32_2_1 stereo stream. Table 7.22 illustrates the effect of different QoS configurations on airtime and latency for bidirectional streams.

QoS Configuration		Airtime (Stereo)	Latency PD = 40ms
Outbound	Inband		
48_2_1	32_2_1	90%	141ms
32_2_1	32_2_1	41%	57ms
24_2_1	16_2_1	31%	56ms

Table 7.22 Airtime and Latency for various bidirectional QoS configurations

It means that there is a trade-off between airtime, QoS and call setup time, which designers need to be aware of. In some applications it may be better to allocate CISes in advance to allow very fast transitions. In others, it could be perfectly acceptable to tear down the CIG and reconfigure it. Although these are primarily use case decisions, based on the resulting user experience, they may also depend on other airtime resource demands in the Initiator. If your device is using Wi-Fi to obtain the music stream you want to listen to, then it has to have enough airtime to do that. It illustrates the flexibility of Bluetooth LE Audio, but highlights the fact that implementors need to understand more of the overall system than they did with classic Bluetooth Audio profiles.

7.7 Who's in charge?

One of the questions for designers is deciding "who is in charge?" The development of Bluetooth LE Audio has seen some strong opinions; from phone manufacturers, who feel that the phone is responsible for everything, but also from speaker and hearables manufacturers who feel that more autonomy needs to be given to their products, leading to the concept of the "Sink led" journey. The specifications give room for both, but they need developers to be aware of some of the consequences.

It's useful to look at a simple example, which is a pair of earbuds, each of which has a microphone. A phone manufacturer might want to use both, as that would allow them to process both audio streams, which should result in a better voice signal. On the other hand, earbud manufacturers might prefer to use only one, conserving the battery life of the other earbud. They might even want to be able to swap which is being used during the course of a call, if they see the battery of the one with the active microphone starting to decline.

The question is, how to enable these options? Assuming there is communication between the earbuds, one of them could consider not exposing its Source ASE, although that's a brute force approach, especially as the phone could infer its presence by reading the PAC records. As we saw above, an Acceptor should expose an ASE for every Audio Stream it is able to support, even if it may not be able to support it at all points in time. A far better method is to indicate that the server is not available to transmit audio by using the Available_Audio_Contexts characteristic in PACS with the Available_Source_Contexts set to 0x0000 (which is one of the few occasions where you are allowed to set all Context Type bits to zero.)

If the Initiator (phone) can see that each earbud has an available Source ASE, then it can effectively take charge. It may decide to select just one (not least because processing two incoming streams which are not time aligned is not trivial and increases its power consumption). It can also use other information, such as checking the battery status of each earbud, to guide its choice of microphone if it only wants to use one.

If it's more intelligent, it could look to see if phrases are regularly repeated during the call, infer that implies poor signal to noise, and add in the second microphone to try to improve the quality by gaining a few dBs of signal. Equally, a phone app could log the duration of calls with specific people, and from that history deduce the likely length of the call, the earbud battery life and use that to inform its decision of how many Audio Streams to set up and the QoS settings that would let the battery survive through the anticipated call duration. So much opportunity for differentiation! I suspect these approaches are unlikely to be rolled out in the short term, as the phone will continue to consider the earbuds as a resource to use as it likes, but looking further ahead, there are many opportunities to make improvements to the user experience.

On the other hand, supporters of the Sink led journey would want more decision making ability in the earbuds. However, that implies an ability for them to communicate with each other, which is currently out of scope (other than the Preset local synchronization in HAPS). If the pair decide that they only want to use the microphone on one earbud, they can decide between them which will be the one to take the battery hit, and the other can set its Available_Source_Contexts set to 0x0000. (They can do this while they're still in the battery box, so this doesn't have to use a separate, sub-GHz radio link to propagate through the head.)

This approach has the advantage that the phone knows that the Source ASE is there for both, so it can configure a stream in the CIS. It can't enable it for the earbud showing Available_Source_Contexts as 0x0000, because the earbud will reject it at the config codec stage. However, the Initiator can schedule it, as it can put whatever it wants in the HCI LE Set CIG Parameters command. Although we've seen that the process normally uses information that the Initiator receives from an Acceptor, it can use cached or assumed values for missing Acceptors or ASEs. That means that if there is a need to swap microphones at some point, that can happen. Otherwise, the Initiator would need to tear down the CIG and rebuild it, resulting in a break in the call. (That shouldn't terminate the call, as the loss of a

Section 7.7 - Who's in charge?

stream doesn't cause any change in the TBS state machine, but it's not a good user experience).

The point here is that the PACS and ASCS characteristics allow a surprising amount of ownership regarding how an Audio Stream is set up. An Initiator can act unilaterally, but that risks affecting how well devices work. For the best performance and user experience, designers need to understand the options offered and how to use them.

--oOo--

That concludes the methods for setting up unicast Audio Streams, so we can now turn our attention to broadcast.

Chapter 8. Setting up and using Broadcast Audio Streams

In this chapter we'll look at how to configure broadcast Audio Streams. Broadcast is the major new feature of Bluetooth® LE Audio and has the potential to change the way that everybody uses audio. The exciting use cases it introduces include the ability to share audio, along with the ability to set up ad-hoc private connections. The latter is enabled by the Broadcast Assistant features which are defined in BASS and bring totally new control and user experiences.

To highlight the importance of broadcast audio, the Bluetooth SIG has developed a new brand name called Auracast™ to promote both its use and the availability of interoperable broadcast audio streams. We'll cover the details of Auracast™ in Chapter 12. First, we need to understand how broadcast works.

Since writing the first edition of this book, it has become clear how challenging a concept broadcast is for many people. For implementers who have several decades of experience with Bluetooth, the concept of devices operating with no connection between them has been extremely difficult. Equally challenging has been the topology, where a third device (or multiple third devices) are used to help find and select the broadcasts, but don't participate in the Audio Stream. To help make it more understandable, I've rearranged this chapter to try and cover the four main features that you need to understand before you start to design and use broadcast audio, They are:

- Transmitting Bluetooth LE Audio streams,
- Receiving Bluetooth LE Audio streams,
- Finding Bluetooth LE Audio streams, and
- Selecting Bluetooth LE Audio streams.

The good news is that you need very few of the Bluetooth LE Audio specifications to do this. The ones required for broadcast are shown in Figure 8.1.

Once again, the main specification is BAP – the Basic Audio Profile. It provides the foundation, telling designers how to transmit and receive broadcast audio streams. Alongside that, we now need to add another one of the GAF specifications called BASS – the Broadcast Audio Scan Service. BASS defines how an additional device can be used to help find broadcast Audio Streams and instruct one or more Acceptors to synchronize with them. With just those two specifications, we can do most of what we need for broadcast. As with unicast, the Published Audio Capabilities Service specification (PACS), helps Acceptors expose their capabilities.

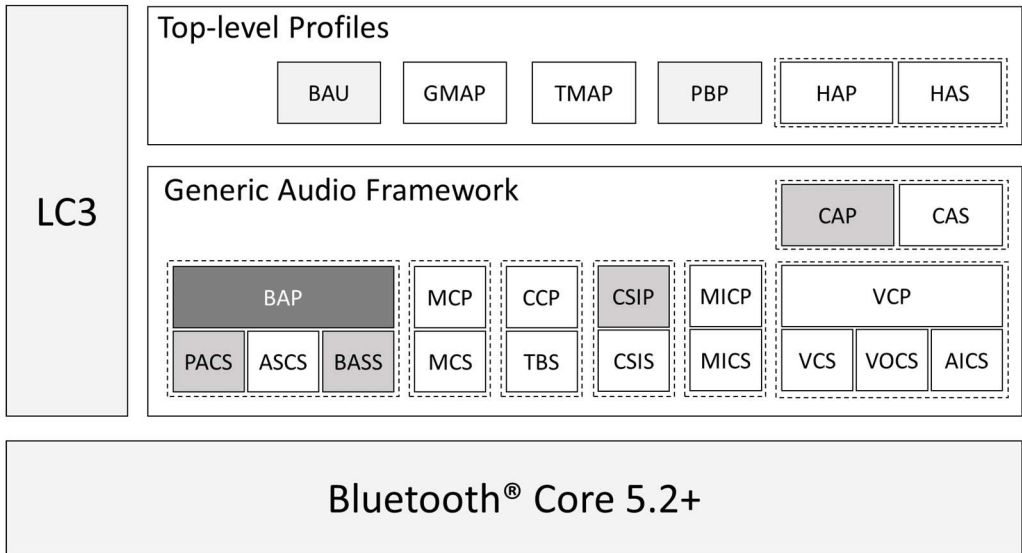


Figure 8.1 The components required for broadcast audio

CAP – the Common Audio Profile, comes into play as it packages the BAP procedures together to set up and receive broadcast streams. It also defines the role of a Commander. The Commander is a role, which may be the only role of a physical device, or implemented as an application on a phone or a TV. We’ll look at the Commander role in more detail once we’ve gone through the basics of sending and receiving broadcast Audio Streams. CAP also lays down rules about associating Context Types and Content Control IDs and confirms the order in which things need to be done.

We’ll start with the basics of setting up and receiving a broadcast Audio Stream, then look at what the Commander brings to the picture. In Chapter 12, we’ll delve further into the use case possibilities within broadcast audio, examining some of the new use cases that it can enable through the Auracast™ experience, where we will see the power of the Public Broadcast Profile (PBP) and the Broadcast Audio URI (BAU).

The broadcast topology of Bluetooth LE Audio originated with a desire to improve the accessibility provided by inductive hearing loops, bringing those features to a wider public. However, it’s not just an alternative to hearing loops - Bluetooth LE Audio provides far more versatility. Although many of the use cases will be truly connectionless, where there is no ACL link between the transmitter and Broadcast Receiver, in many cases there will be. A Broadcast Source and the Broadcast Receiver can communicate with each other to supply encryption information, or even to automatically detect the presence of earbuds and start streaming automatically. Devices can switch back and forth between unicast and broadcast, and support both at the same time, either to mix audio streams or to act as a relay between broadcast and unicast connections. But before we get into those complications, we’ll start with the key

Section 8.1 - Setting up a Broadcast Source

elements of broadcast by itself.

The broadcast specifications define three separate functions:

- Transmitting a broadcast Audio Stream. At its simplest, a Broadcast Transmitter acts independently of other devices – it generally has no idea whether any receivers are present and listening to it.
- Finding a broadcast Audio Stream. This will generally use a Broadcast Assistant, often referred to as a Commander. Technically, a Commander is just a role, of which the Broadcast Assistant is a sub-role, but the industry has adopted the name of Broadcast Assistant as a user friendly way of describing an app or a device which finds or selects a Broadcast. A Broadcast Assistant can find broadcast Audio Streams for a Broadcast Receiver. Broadcast Assistants elevate broadcast from being a simple telecoil replacement into a very powerful new topology, which allows encrypted streams to be used for private audio, both at a personal and an infrastructure level. They also make it much easier to select amongst multiple broadcast Audio Streams. Broadcast Assistants can be designed as stand-alone devices, or can be collocated with any Broadcast Source.
- Receiving a Broadcast Audio Stream. A Broadcast Receiver, (which is essentially the same as a Broadcast Sink), can scan for the presence of a broadcast Audio Stream and synchronize to it. At this basic level, it acts independently. A Broadcast Receiver can synchronize to encrypted or unencrypted broadcast Audio Streams, but needs to obtain the Broadcast_Code to decrypt an encrypted Audio Stream. This can be done out of band, or with the help of a Broadcast Assistant

8.1 Setting up a Broadcast Source

In Chapter 4 we covered the basics of Broadcast Isochronous Streams (BIS) and Broadcast Isochronous Groups (BIG). Unlike unicast streams, a Broadcast Source and Broadcast Sink operate independently. This makes the Broadcast Source very different from any other Bluetooth Central device, as it acts unilaterally. Unlike the unicast case, which we explored in the previous chapter, no commands, requests or notifications take place between the Source and the Sink. Instead, the Broadcast Source is driven entirely by its specific application.

One consequence of this is that a Broadcast Source has a very simple state machine, shown in Figure 8.2. As there are no interactions with any Acceptors, the procedures are very straightforward, consisting of commands from the Host to the Controller within the Broadcast Source.

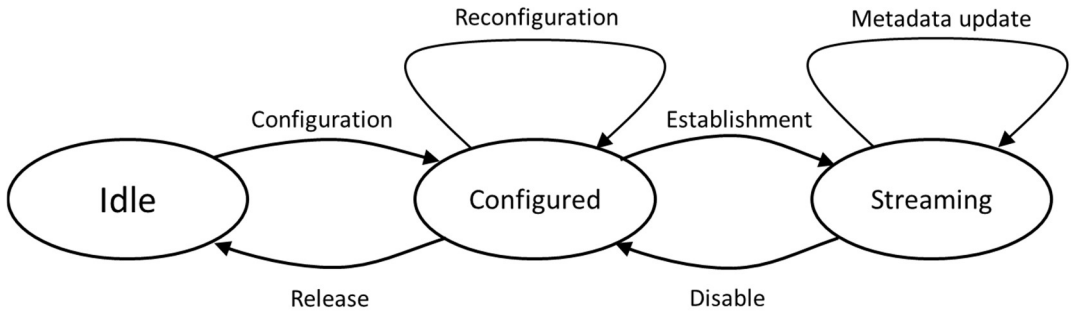


Figure 8.2 Broadcast Source State Machine and Configuration procedures

To configure a Broadcast Source, the Host needs to provide the Controller with details of the BIG configuration. This is used by the Controller to schedule the BISes. The Host also needs to provide the controller with information to populate the BASE, which describes the configuration of each BIS and its content. The configuration data is provided by the application running the Broadcast Source. In some applications, the metadata for inclusion in the BASE may be part of an application; in others it might be supplied externally, either from a control input, or extracted from an incoming audio stream, such as a TV's electronic program guide data. Some of the metadata is configurable by the user, such as the device's name.

BAP defines six procedures for transitions and configuration updates of a Broadcast stream:

- The Broadcast Audio Stream configuration procedure
- The Broadcast Audio Stream establishment procedure
- The Broadcast Audio Stream disable procedure
- The Broadcast Audio Stream Metadata update procedure
- The Broadcast Audio Stream release procedure, and the
- The Broadcast Audio Stream reconfiguration procedure

Because there are no connections between devices, these procedures are mostly confined to HCI commands, sent from the Initiator's Host to its Controller when it is operational. CAP bundles them together into just three procedures for the Broadcast Transmitter, combining configuration and establishment into its Broadcast Audio Start procedure, giving us the:

- Broadcast Audio Start procedure
- Broadcast Audio Update procedure
- Broadcast Audio Stop procedure

What surprises most people is that there are no procedures defined for an Acceptor to synchronize to a Broadcast Stream by itself. It can certainly do it, and I'll describe that process later in the chapter, but the assumption made within Bluetooth LE Audio is that it will always use a Broadcast Assistant to do the job for it. There are procedures defined for how this is

Section 8.2 - Starting a broadcast Audio Stream

done, with CAP defining two of them:

- Broadcast Audio Reception Start procedure
- Broadcast Audio Reception Stop procedure

CAP once again brings in support for Coordinated Sets, which is vitally important, as a pair of Acceptors may have no knowledge about each other⁵⁸.

8.2 Starting a broadcast Audio Stream

As the Broadcast Source has no knowledge of what might be receiving its broadcast Audio Streams, none of the CAP preamble procedures concerning Coordinated Sets are relevant. All CAP requires is that the correct Context Types are included in the metadata. Content Control IDs are only required if there is an accompanying ACL connection carrying content control from the Broadcast Source. This would be required in applications like a personal TV, which uses broadcast to allow multiple family members to listen, but where their individual earbuds have an ACL connection that they can use to pause or change channel.

Everything else we need is defined in BAP, where the procedures instruct the Controller to set up Extended Advertising and provide the data to populate the parameters which are exposed in the Extended Advertisements and the Periodic Advertising train.

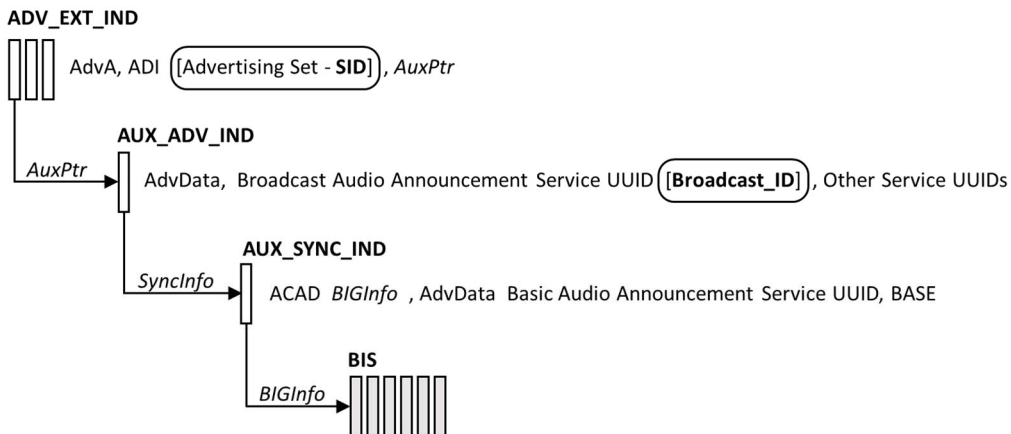


Figure 8.3 Data required to set up a Broadcast Source

Figure 8.3 takes the Extended Advertising diagram from Chapter 4 and highlights the specific elements of data required to set up a Broadcast Source. The first step in setting up a broadcast

⁵⁸ In practice, they normally do know about each other, as most manufacturers include sub-GHz or NFMI radios in earbuds and hearing aids to let them talk to each other. But the Bluetooth LE Audio specifications do not require this. It is less likely that pairs of speakers will have these additional links, so they will need to rely on CAP.

Audio Stream is to assemble all of the data needed for the Controller to populate the Broadcast Audio Announcement Service, BIGInfo and BASE, the details of which are described in the BAP Audio Stream Establishment procedure [BAP 6.3].

8.2.1 Configuring the BASE

The Application will determine how many BISes are going to be transmitted and their associated codec and QoS configurations. BAP recommends that at least one of the broadcast Audio Streams should be encoded with either the 16_2 or 24_2 settings of Table 3.2, i.e., 16kHz, 10ms SDU or 24kHz, 10ms SDU, to ensure that every Acceptor can decode it. Any other additional codec configurations will be determined by the application or a higher level profile. The Host should also obtain any metadata content which it requires to populate the BASE. Current metadata requirements are shown in Table 8.1.

Profile	Required LTV metadata structures	Comments
BAP	None	
CAP	Streaming_Audio_Contexts	
CAP	CCID_List	Only if content control exists for the Audio Stream
HAP	None	
TMAP	None	
PBP	None	The Broadcast_Name LTV is recommended for Auracast™ compliant applications. The ProgramInfo LTV is recommended to describe the content of the Audio Stream.

Table 8.1 Metadata LTV requirements for BASE from different profiles

The final piece of information needed to configure the stream is the value of Presentation Delay, which is generally determined by the top level profile.

The Controller can now put together its BASE structure, which describes exactly what streams it will be transmitting and their configuration.

We went through the overview of the BASE in Chapter 4, looking at what it contains, which is repeated in Figure 8.4, emphasising the three levels, which contain global BIG parameters (Level 1), common subgroup parameters (Level 2) and individual BIS parameters (Level 3).

Level 2 parameters are arrayed by subgroup. Level 3 parameters are arrayed by the BIS within each subgroup.

Section 8.2 - Starting a broadcast Audio Stream

8.2.1.1 Subgroups in BASE

Information on how to split BISes into subgroups was very vague in the 1.0.0 and 1.0.1 versions of BASE, but has been clarified in the latest 1.0.2 release. It is recommended a subgroup should only include BISes which are intended to be rendered in the same, or the same set of devices.

Some of these are obvious. A stereo set of left and right BISes would be included in the same group. A corresponding set of left and right in a different language would be in a separate subgroup, as it is unlikely that a user would want to render any combination of these in the same group.

Other decisions are a little more subtle. If a Broadcast Source transmits left, right and mono, these would not normally be rendered together in the same set of Broadcast Sinks, so mono would be placed in its own, separate subgroup, as shown in Figure 8.4. However, for an infrastructure application, such as a theatre, where left, right and mono were transmitted for multiple, different speakers, it might be valid to include all three in one subgroup. Product designers should think carefully about the intended application and the capabilities of the anticipated Broadcast Sinks which will be interpreting the BASE content.

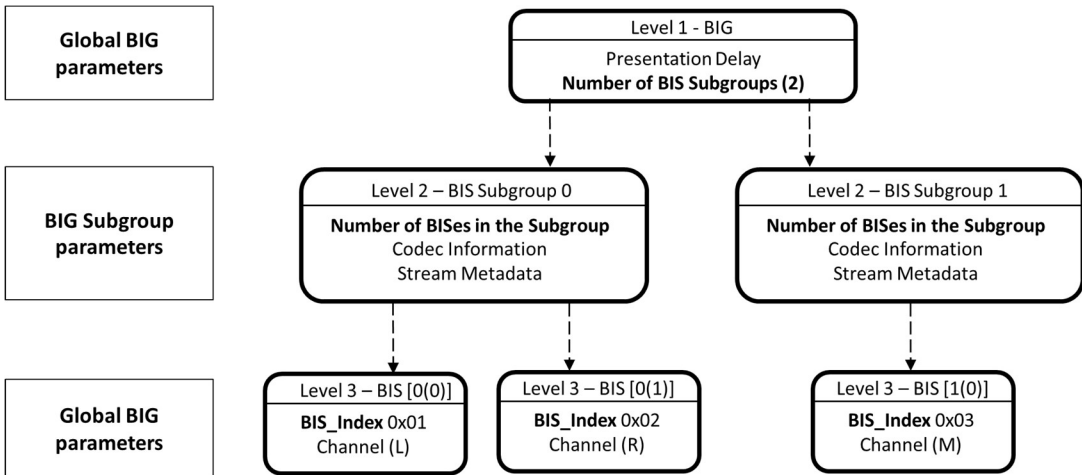


Figure 8.4 Simplified BASE structure

Moving back to the BASE, the entire BASE is an LTV structure which is presented as an AD Type, with the parameters shown in Table 8.2.

Parameter	Size (Octets)	Description
Length	1	Total length of the LTV structure.
Type	1	Service-Data UUID
Level 1 - BIG Parameters (common to all BISes)		
Basic Audio Announcement Service UUID	2	0x1852 (from Bluetooth Assigned Numbers).
Presentation Delay	3	Range from 0 to 16.7 sec in μ s (0x000000 to 0xFFFF)
Num_Subgroups	1	The number of subgroups [i] used to group BISes with common features in this BIG.
Level 2 – BIS Subgroup Parameters (common parameters for subgroups of BISes)		
Num_BIS[i]	1	The number of BISes in subgroup [i].
Codec_ID[i]	5	Codec information for this subgroup. Typically, this will be LC3 (0x0000000006).
Codec_Specific_Configuration_Length[i]	1	The length of the codec configuration data for the Codec_ID in this subgroup.
Codec_Specific_Configuration[i]	varies	The codec configuration data for the Codec_ID in this subgroup.
Metadata_Length[i]	1	Length of the metadata for this subgroup
Metadata[i]	varies	The metadata for this subgroup. CAP requires the Streaming_Audio_Contexts LTV Metadata for every subgroup, and also the CCID_List LTV structure if content control is applied.

Section 8.2 - Starting a broadcast Audio Stream

Parameter	Size (Octets)	Description
Level 3 – Specific BIS Parameters (if required, for individual BISes)		
BIS_index[i[k]] ⁵⁹	1	Unique index for each BIS in the BIG.
Codec_Specific_Configuration_Length[i[k]]	1	The length of the codec configuration data for a specific BIS [i[k]] in this subgroup.
Codec_Specific_Configuration[i[k]]	varies	The codec configuration data for the specific BIS [i[k]] in this subgroup. This generally contains the Audio_Channel_Allocation LTV

Table 8.2 The parameters of the BASE.

The BASE allows a lot of flexibility, the majority of which will not be needed in most everyday applications. The Level 1 parameters must exist and apply to every subgroup. Most BIGs will probably only have a single subgroup, which will typically carry two or three BISes – one for left and one for right, plus an optional mono stream for hearing impaired users⁶⁰. The reason for defining more than one subgroup is for occasions when some of the BISes have different metadata, such as different languages. Differences like this can only be expressed in Level 2 parameters. At Level 3, you can specify different codec configurations for BISes within a subgroup, such as having different quality levels, for example one stream at 48 kHz sampling and one at 24 kHz⁶¹. But that is the only thing which changes at Level 3. The Level 3 also contains the Audio_Channel_Allocation LTV which specifies the intended Sink Audio Location for the stream. Different languages, different parental ratings, etc., all need to have their own subgroup defined at Level 2.

⁵⁹ One point of possible confusion that developers need to be aware of is that arrayed indices start from 0, but BIS numbering starts from 1. In the example of Table 8.3, the first BIS in subgroup 1 is represented by the BIS_index[0,[0]], and has a BIS_index value of 0x01,

⁶⁰ For the last fifty years, most feature films shown in cinemas have included a “dialogue boost” or “assisted listening” audio track for hearing impaired listeners, where background noises and music are reduced to make speech easier to interpret. This feature is increasingly being included in TV and video streaming services. Just check the language options for Amazon.

⁶¹ Remember that if difference codec sampling rates are chosen for different BISes, the BIG will be configured so that all BISes accommodate the size of the largest packet. It is usually more effective to set up multiple BIGs, each with their own BASE.

Where there is more than a single BIS, the Level 2 and Level 3 parameter sections should be repeated for each BIS in order of subgroup and BIS_index, as shown in Table 8.3, which shows values for the example of Figure 8.4.

Level	Parameter	Value
Subgroup 1		
2	Num_BIS [0]	Num_BIS = 0x02
2	Codec_ID [0]	LC3 = 0x06
2	Codec_Specific_Configuration Length [0]	0x0A octets
2	Codec_Specific_Configuration [0]	LTV1 Sampling Frequency - 24kHz LTV2 Frame Duration – 10ms LTV3 Octets per codec Frame - 60
2	Metadata Length [0]	0x09
2	Metadata [0]	LTV1 Streaming_Audio_Context: Media LTV2 Language - English
3	BIS_index [0[0]]	0x01
3	Codec_Specific_Configuration_Length [0,[0]]	0x06
3	Codec_Specific_Configuration [0[1]]	LTV1 Audio_Channel_Allocation FL
3	BIS_index [0[0]]	0x02
	Codec_Specific_Configuration_Length [0,[1]]	0x06
3	Codec_Specific_Configuration [0[1]]	LTV1 Audio_Channel_Allocation FR
Subgroup 2		
2	Num_BIS [1]	Num_BIS = 0x02
2	Codec_ID [1]	LC3 = 0x06
2	Codec_Specific_Configuration Length [1]	0x0A octets
2	Codec_Specific_Configuration [1]	LTV1 Sampling Frequency - 24kHz LTV2 Frame Duration – 10ms LTV3 Octets per codec Frame - 60
2	Metadata Length [1]	0x09
2	Metadata [1]	LTV1 Streaming_Audio_Context: Media LTV2 Language - German
3	BIS_index [1[0]]	0x03
3	Codec_Specific_Configuration_Length [1[0]]	0x06
3	Codec_Specific_Configuration [1[0]]	LTV1 Audio_Channel_Allocation FL

Section 8.2 - Starting a broadcast Audio Stream

Level	Parameter	Value
3	BIS_index [1[1]]	0x04
	Codec_Specific_Configuration_Length [1[1]]	0x06
3	Codec_Specific_Configuration [1[1]]	LTV1 Audio_Channel_Allocation FR

Table 8.3 Order of entries for multiple subgroups and BISes

8.2.2 Creating a BIG

As we saw with unicast, the BIG is created using an HCI command – in this case the LE_Create_BIG command, with the parameters shown in Table 8.4.

Parameter	Description
BIG_Handle	The BIG identifier, set by the Host.
Advertising_Handle	The handle of the associated periodic advertising train.
Num_BIS ⁶²	The number of BISes in the BIG.
SDU_Interval	The interval between the periodic SDUs containing encoded audio data in microseconds.
Max_SDU	The maximum size of each SDU, in octets.
Max_Transport_Latency	The maximum transport latency for each BIS, including pre-transmissions.
RTN	The requested number of retransmissions (which is a recommendation)
PHY	A bitfield of acceptable PHY values. The Controller will choose from one of the supported options. A high-level profile may mandate only one value – normally 2Mbps. 0 = 1Mbps, 1 = 2 Mbps, 2 = LE coded.
Packing	The preferred method of arranging BIS Subevents. The Controller only uses this as a recommendation. 0 = Sequential, 1 = Interleaved.
Framing	If set to 1, the BIS data PDUs will be framed. If set to 0, the Controller can decide whether to use framed or unframed.
Encryption	Set to 1 if the audio stream needs to be encrypted and 0 for unencrypted.
Broadcast_Code	The code used to generate the encryption key for all BISes in the BIG. This should be set to zero for an unencrypted stream.

Table 8.4 Parameters for the HCI LE Create BIG command

⁶² Note that there is a parameter in the Core with the same name of Num_BIS, but which has a different meaning.

An important difference between the LE Create BIG Parameters command and the equivalent LE Set CIG Parameters command, is that the same parameters are used across every BIS in the BIG. That limitation arises from the fact that the Isochronous Channel information is contained in the BIGInfo packet, which is not large enough to accommodate details of multiple, different BISes. The consequence is that the size and number of subevents for every BIS are the same size, hence it must be long enough to fit the largest SDU. Different BISes within the BIG can use different codec configurations and even different codecs, but the BIS structure must be configured to fit the largest possible packet across those different configurations. If different QoS settings are used for different BISes in the BIG, it means that there will be redundant space and wasted airtime. If this is the case, it is often more efficient to separate the BISes into different BIGs, with each sized to optimise airtime. Although they could use the same value for Presentation Delay, they would be rendered at slightly different times because they will have different BIG Anchor Points.

Having created the BASE structure, the Host can ask the Controller to schedule the BIG and BISes, using the LE_Create_BIG command [Core Vol 4, Part E, Section 7.8.103], with the parameters shown in Table 8.4.

Once the command has been issued and the Controller has worked out its scheduling, it will respond to the Host with an LE_Create_BIG_Complete event [Core Vol 4, Part E, Sect 7.7.65.27] containing the actual parameters which it has chosen to use, as shown in Table 8.5. Some of these are used by the Host application, others will be used to populate the BIGInfo – the data structure which informs scanning devices of the configuration of the broadcast Isochronous Streams.

Parameter	Description	Used in
Subevent Code	0x1B – Subevent code for the LE_Create_BIG_Complete event	Host
Status	Set to 0 if the BIG could be successfully scheduled; otherwise set to 1, when an Error Code is available.	Host
BIG_Handle	The same BIG handle the Host sent in its LE_Create_BIG command	Host
BIG_Sync_Delay	The maximum time for transmission, in microseconds, of all PDUs of all BISes in a single BIG event, using the actual parameters included below.	Host
Transport_Latency_BIG	The actual transport latency for the BIG in microseconds. (This covers all BIG events used for an SDU.)	Host
PHY	The PHY that will be used for transmission.	BIGInfo
NSE	The number of Subevents for each BIS	BIGInfo

Section 8.2 - Starting a broadcast Audio Stream

Parameter	Description	Used in
BN	The Burst Number. (The number of new payloads in each BIS event.)	BIGInfo
PTO	The offset used for pre-transmissions.	BIGInfo
IRC	The Immediate Repetition Count. (The number of times a payload is transmitted in each BIS event.)	BIGInfo
Max_PDU	The maximum size of the payload in octets.	BIGInfo
ISO_Interval	The value of the Isochronous Interval. (The time between consecutive BIG Anchor Points.) Expressed as a multiple of 1.25 msec.	BIGInfo
Num_BIS	The total number of BISes in the BIG	BIGInfo
Connection Handle[i]	A list of Connection Handles for all of the BISes in the BIG	Host

Table 8.5 Parameters returned by an LE_Create_BIG_Complete HCI event

At this point, the Host has everything it needs to start the process, the first stage of which is to set up the Extended Advertising and Periodic Advertising trains. It needs to include the Broadcast Audio Announcements [BAP 3.7.2.1] and Broadcast_ID [BAP 3.7.2.1.1] in the AUX_ADV_IND Extended Announcements, and the Basic Audio Announcement, including the BASE [BAP 3.7.2.2] and BIGInfo

The Host uses the HCI_LE_Set_Extended_Advertising_Data command [Core Vol 4, Part E, Sect 7.8.54] and enters the Broadcast mode [Core Vol 4, Part C, Sect 9.1.1] to start the Extended Advertising, then uses the HCI_LE_Set_Periodic_Advertising_Data command [Core Vol 4, Part E, Sect 7.8.62] to populate the BASE, before entering the Periodic Advertising Mode [Core Vol 4, Part C, Sect 9.5.2].

Once the Extended Advertisements and the Periodic Advertising train are established, the Broadcast Source enters the Configured State. At this stage it is not yet transmitting any audio data. As a Broadcast Source knows nothing about which Acceptors will attempt to synchronize to its streams, it does not need to follow any of the CAP procedures for Coordinated Sets in the way that a Unicast Source would need to.

8.2.3 Selecting the Presentation Delay value

With unicast, Acceptors let the Initiator know about their ability to support Presentation Delay by exposing their Maximum and Minimum Presentation Delay capabilities, as well as their preferred value range. In broadcast, there is no information transfer between Initiator and Acceptor. This means that the value which an Initiator sets may not be supported by all of the Acceptors which decide to receive it.

BAP requires that all Broadcast Receivers must support a value of 40ms in their range of supported Presentation Delays, so this is probably a default value which many Initiators will

use. However, it increases the overall latency, which may be excessive for live audio.

TMAP and HAP place tighter constraints on the value of Presentation Delay, mandating that an Acceptor must support any value from 20ms to 40ms. As most Acceptors are expected to be qualified to TMAP or HAP (most will probably support both), setting the lowest value of 20ms for a Broadcast Source seems a reasonable compromise. However, a Broadcast Source which is only BAP compliant may not support this.

This raises the question of what an Acceptor should do if it does not support the value of Presentation Delay exposed by a Broadcast Source. The specification is silent on this, but the following guidelines offer a pragmatic approach.

- If the Presentation Delay in the BASE is lower than an Acceptor can accommodate, the Acceptor should use the lowest value it can support, which must be at least 40ms (as required by BAP).
- If the Presentation Delay in the BASE is higher than an Acceptor can support, it should use the highest value it can support, which must be at least 40ms (as required by BAP).

There is an assumption in both of these statements that an Acceptor will support any value between its minimum and maximum values and not just “spot” values of 20ms and 40ms.

The specification gives a Broadcast Source three octets to carry the Presentation Delay value. That gets over the limitation of 65ms, which would be the maximum if only two octets were used (the units are 1µsec), but it means that the value could be as high as 16.7 seconds, although it's unlikely it would ever be set that high. Most Acceptors will have limited memory for buffering the audio stream, so there may be occasions where an inappropriately high value of Presentation Delay is outside their capability. Whilst they could decide not to render the Audio Stream, the pragmatic approach is to revert to the BAP value of 40ms.

In both of these cases, this should help to ensure that both left and right devices render at the same time. If they are able to communicate with each other, they may use a proprietary method to determine the most appropriate value, which will generally select the lowest common value they support.

A further option is for a Broadcast Source to include the Broadcast Audio Immediate Rendering Flag (BAIRF) LTV [PBP 5.4] in its BASE metadata, or its Public Broadcast Announcement metadata. This instructs an Acceptor to render the audio stream as quickly as they can. If a pair of earbuds or hearing aids want to act on this, they need to have some method of informing each other, so that both make the same decision. How this is done is left to implementation. Headphones would automatically apply the same latency to each output transducer. With speakers, a method of communication is still useful, but the greater physical distance between them would probably mean that in most cases it would not be

Section 8.2 - Starting a broadcast Audio Stream

noticed if only one applied the faster rendering.

8.2.4 Updating a broadcast Audio Stream

At any point, a Broadcast Source can update the LTV structures in the BASE containing the metadata. This is normally done to reflect changes in the Audio Channel content, such as new ProgramInfo or Context Types. This is defined in the CAP Broadcast Audio Update procedure, which references the BAP Modifying Broadcast Sources procedure [BAP 6.5.5]. The Broadcast Source does not need to stop the Audio Stream, but can make the change dynamically when in the Streaming State, which will update the content of the BASE.

There is no guarantee that an Acceptor which is receiving the Audio Stream will see such a change. Once an Acceptor has synchronized to a stream using the information in the BIGInfo, it has no further need to track the Periodic Advertisements or read the BASE, unless required to do so by another profile. At the moment, no profile contains this requirement.

The decision for an Acceptor to look for updates is an implementation choice, as it will affect its power budget. A Broadcast Source cannot assume that any Acceptors are continuing to track the Periodic Advertisements.

The only other feature which can be changed once a BIG is established is the hopping sequence. This is changed within the BIGInfo. To prevent Acceptors losing synchronization, these updates are signalled using Control Subevents (see Section 4.4.3).

8.2.5 Establishing a broadcast Audio Stream

Once the Extended and Periodic Advertising is running, the Broadcast Source needs to establish its Audio Streams and start transmitting. It does this in three steps, defined by the BAP Broadcast Audio Stream Establishment⁶³ procedure [BAP 6.3.2].

First, the Broadcast Source enters the Broadcast Isochronous Broadcasting Mode [Core Vol 3, Part C, Sect 9.6.2], which prepares it to send PDUs in the BISEs of its BIG. It then starts the Broadcast Isochronous Synchronizability Mode [Core Vol 3, Part C, Sect 9.6.3], which starts sending the BIGInfo in the ACAD fields of the Periodic Advertisement, alerting any devices that are scanning for broadcast Audio Stream to its presence.

Finally, the Broadcast Source needs to set up the audio data path in the same way as for unicast, using the LE Setup ISO Data Path HCI command [Core Vol 4, Part E, Sect 7.8.109].

At this point, the Broadcast Source is fully operational, transmitting its BIG and constituent BISEs. It now needs to ensure that it configures the BASE, if it has not already done so, and

⁶³ This should not be confused with the BASE. It's unfortunate it has the same initials.

if it is Auracast™ compliant, enable the Public Broadcast Announcement [PBP Sect 4]. If it has no connections to any Broadcast Sinks, which will often be the case, it will never know whether any device detects or synchronizes to those broadcasts.

8.2.6 Stopping a broadcast Audio Stream

To stop broadcasting, and return to the Configured state, an Initiator needs to use the Broadcast Isochronous Terminate procedure [Core Vol 3, Part C, Sect 9.6.5]. This stops the transmission of BIS PDUs, the BIGInfo, the BASE and Public Broadcast Announcement. Synchronized Acceptors will receive a BIG_TERMINATE_IND PDU to alert them to the end of transmission. If they fail to receive this, they get no further information other than the fact that both the BIS and BIGInfo have disappeared, at which point they should use a local timeout to indicate that they are no longer synchronized to the BIS or the PA. It is important to stop the BIG correctly, so that any Acceptors which are synchronized to one of its stream are aware that the stream has been terminated. Otherwise, they are likely to waste power attempting to find the missing Broadcast Source.

At the end of the Broadcast Isochronous Terminate procedure, the Broadcast Source will return to the Configured State. It can then be released, or re-enabled.

8.2.7 Releasing a broadcast Audio Stream

To return a Broadcast Source to its Idle state, it needs to exit the Periodic Advertising mode, which returns it to the Idle state.

8.3 Receiving broadcast Audio Streams

From this point on, I'll mostly use the terms Broadcast Source (or Transmitter), Broadcast Sink (or Receiver) and Broadcast Assistant, as they are becoming the de facto terms used in discussing broadcast applications.

If a Broadcast Sink wants to receive a broadcast Audio Stream, it needs to scan to find the Broadcast Source. The same procedures for finding the Broadcast Source are used, whether this is being done by a Broadcast Sink itself, or a Broadcast Assistant. We'll just look at the Broadcast Sink doing it in this section, then see how the task can be delegated when a Broadcast Assistant is available.

Because Acceptors act as separate entities, CAP doesn't really come into play for the reception of broadcast Audio Streams. CAP defines a Broadcast Audio Reception Start procedure [CAP 7.3.1.8], but unless the Acceptors have a way of communicating with each other, they don't know that the other is present. They know they're a member of a Coordinated Set, but not whether the other set members are around. A Broadcast Assistant can perform that coordination task, which we'll come to later, or there could be an out of band mechanism, such as a sub-GHz radio which allows the Acceptors to talk to each other. But a Broadcast Sink scanning on its own is essentially operating independently at the BAP level.

Section 8.3 - Receiving broadcast Audio Streams

8.3.1 Audio Announcement discovery

The first step in receiving a broadcast stream is to find it, which is performed using the Observation Procedure from the Core [Core Vol3, Part C, Sect 9.1.2]. This is a standard scan to find advertisements. Here, the advertisements of interest are Extended Advertisements which contain the Broadcast Audio Announcement Service UUID with a Broadcast_ID. If the transmission is Auracast™ compliant it will also contain the Public Broadcast Announcement. The scanner can use the information in the Public Broadcast Announcement to filter the streams which it wants to investigate.

Once it has discovered these, it can decide whether to proceed. If it is interested in the stream based on the information it has found in the Announcements in the Extended Advertisement, the scanner will look for the SyncInfo data and use this to synchronize to the Periodic Advertisements associated with the Broadcast_ID. Once synchronized, the scanner can find and parse the data in the BASE, which provides information about the content and format of the BISes.

At this point, the Acceptor is acting quite differently from previous Bluetooth peripherals. Instead of being told what to do by a Central device, it is collecting information by and for itself, generating a list of the available Broadcast Sources within range which have Audio Streams of interest before making a decision about which one to receive. The first decision it may make is whether to look for more broadcast Audio Sources, or just proceed with the first one it finds. Both are perfectly valid approaches – it's up to the implementation to decide which one to take. That may depend on the form factor, i.e. whether it has a user interface that can easily cope with choosing between multiple streams.

A Broadcast Assistant, such as an application on a phone, may decide to find every Broadcast Source and present those to the user, to select the one they want. In contrast, a hearing aid may select the first one it finds, with the user pressing a button to find the next available source if the first one was not the one they wanted. In both cases the implementation may employ filtering, such as prioritising a known Broadcast_ID, which it has connected to before, or a particular audio quality, (which is exposed in the Public Broadcast Announcement), or a specific language. All of that is implementation specific.

As more Broadcast Transmitters are deployed, devices will need to adopt more filtering options to ensure a good user experience. These are likely to be based on the codec configuration data and metadata LTVs exposed in the BASE. For example, a Broadcast Sink would not normally bother with streams if their Context Type did not match any of the Acceptor's Available Context Types or its supported codec configuration⁶⁴. Nor would it

⁶⁴ Devices which need extended battery life, such as hearing aids, may conserve power by not supporting or accepting LC3 sampling rates above 24kHz.

want to receive a stream for an Audio Location that it does not support, or a language which its user does not understand. But the decision is the Broadcast Sink's.

Without a Commander, or the presence of a proprietary link between two Broadcast Sinks, a user might have to choose a stream manually on both their left and right earbuds. It is unlikely that this will ever be the case, as it's such a bad user experience; manufacturers will either provide a separate Broadcast Assistant (or a Broadcast Assistant application), or implement a proprietary link between left and right earbuds. For that reason, designers should be aware of the need to allow pairs of devices to work together when they do not have a Bluetooth link between them. But that's where the remote control / Broadcast Assistant comes in. Before we discuss that, we'll consider the process of synchronizing to a broadcast Audio Stream. Once again, we'll look at the case of a single Acceptor.

8.3.2 Synchronizing to a broadcast Audio Stream

Once it has decided on a Broadcast Source it wants to receive, a Broadcast Sink starts the Broadcast Audio Stream synchronization procedure [BAP 6.6]. This involves it reading the BASE information from the AUX_SYNC_IND (and any supplementary AUX_CHAIN_IND) PDUs to determine the codec configuration and the index number of the BIS which corresponds to the Broadcast Sink's Audio Location (See Section 8.3.4). The BIS_Index for each BIS defines the order of the BISes within the BIG. Knowing this, the Acceptor can determine which BIS or BISes in the BIG it wants to receive.

The Broadcast Sink then retrieves the BIGInfo, which contains all of the information of the BIG structure, the hopping sequence and where the BIS Anchor Points are located. Once it has this, it can invoke the Broadcast Isochronous Synchronization Establishment procedure from the Core [Vol 3, Part C, Sect 9.6.3] to acquire the appropriate BIS. If it has not already done so, it needs to set up its Audio data path. Having received the incoming audio packets, it then applies the Presentation Delay value to determine the rendering point.

8.3.3 Stopping synchronization to a broadcast Audio Stream

If a Broadcast Sink decides it wants to stop receiving a broadcast Audio Stream it acts autonomously to disconnect the Audio data path and stops synchronization with the PA and the BIS.

8.3.4 Using Audio Locations to select the correct audio stream

The use of Audio Locations has a few subtle differences from unicast. In unicast, the Sink Audio Locations characteristic value tells the Initiator what each Acceptor expects – it is essentially stating the physical location of a speaker or earbud. As such, it guides the Initiator in setting up the CISes, which are audio pipes, but the Initiator can send whatever audio content it wants down those pipes. For example, its application may decide to send a left audio stream to a right earbud.

In broadcast, the Audio_Channel_Allocation in Level 3 of a subgroup is a statement of the

Section 8.4 - The broadcast reception user experience

content of a BIS. It allows a Broadcast Assistant to select a specific BIS for each of its Broadcast Sinks, or a Broadcast Sink to make a unilateral decision about which stream it wants to receive. In both cases, the scanning device knows the location related content of each BIS and makes decisions on which BIS to render that are independent of the Broadcast Transmitter.

Note that the way a mono stream is signalled has changed between the initial versions of BAP (1.0 and 1.0.1) and the latest version (BAP 1.0.2). Originally, a Broadcast Sink configured to render mono was not expected to expose a Sink Audio Location characteristic. From BAP 1.0.2, that has been clarified. If a Broadcast Sink only supports mono streams, the Sink Audio Location characteristic should be omitted. However, if it can accept other Audio Locations, either from a local action, or allowing the Sink Audio Location characteristic to be written, then the characteristic should be present and set to set to 0x00000000 when mono is the desired location. This resolves a problem in the earlier versions where a characteristic would need to be removed or created when changing the location preference of a speaker.

A Broadcast Source should generally place a mono stream in a separate subgroup from one carrying a left and right pair of BISes, and not include an Audio_Channel_Allocation LTV for the mono BIS. This follows the rule that was explained in Section 8.2.1.1, where the BISes within a subgroup should be limited to those that are expected to be rendered by a set of devices. In most instances, a mono stream would not be rendered by the same earbuds or speakers as the stereo stream.

Developers should be aware of these changes and design their Broadcast Sinks and Assistants to accommodate any differences in earlier Broadcast Sources.

8.4 The broadcast reception user experience

Although an Acceptor is allowed to acquire a Broadcast Stream by itself, it may not provide a very good user experience. At its most basic, it's an exact analogue of the telecoil experience, where someone wearing a hearing aid presses its telecoil button to acquire a telecoil stream. That user experience works because telecoils are inductive loops and you are normally only within range of one loop, which means there's no issue of whether or not you're connecting to the right one. If there is only one Bluetooth LE Audio transmitter within range, that experience will be the same, but as soon as broadcast applications become popular, that's no longer going to be true. In many cases you're likely to be within range of multiple Broadcast Transmitters. It's exactly the same situation as we have with Wi-Fi. If you scan for Wi-Fi access points with your phone or laptop, you'll frequently find a dozen or more. The user experience for Bluetooth LE Audio is potentially more difficult, because in most cases you'll be wearing a pair of earbuds or hearing aids, which have a minimal user interface. There's also the added complication of needing to start and stop streaming to both members of a Coordinated Set at the same time, and always connecting both earbuds to the same Broadcast Source.

So, although the explanations above regarding how you find a Broadcast Source are valid, they're largely academic as the resulting user experience is not a good one. To make this work in the real world we need to involve another device which does a better job of the hard work of finding broadcast streams and telling one or more Broadcast Sinks what to do with them. We also need a way to distribute the keys needed to decode encrypted Audio Streams. Which brings us to Commanders and Broadcast Assistants.

8.5 Commanders as Broadcast Assistants

Although most people see broadcast audio as the big new feature of Bluetooth LE Audio, the most important innovation is probably the concept of the Commander, which, in the form of a Broadcast Assistant provides remote control and management of broadcast Audio Streams. In a new world of multiple broadcast streams, these Broadcast Assistants provide a simple way for users to select what they hear. Although the concept of broadcast was inspired by the telecoil experience of picking up inductive loops in hearing aids, Bluetooth LE Audio broadcast applications go a long, long way beyond what telecoil can do, not least because of the power of the Broadcast Assistant.

Looking back at the inductive loop paradigm, this was straightforward – you would only receive an audio stream if you were standing within the boundary of the loop. (There might be a problem if you were in the room directly above it, but that was an edge case.) With Bluetooth broadcasts, they can and will overlap and go through walls, so users need a simple way to select the broadcast they want to listen to. We've already seen how the metadata information in BASE and service information from profiles like the Public Broadcast Profile (PBP) help inform that choice. However, that is just for public broadcasts, which are open for anyone to hear. For personal broadcasts, the broadcast Audio Streams would normally be encrypted, requiring methods to acquire the encryption keys. Once again, that is enabled by Broadcast Assistants.

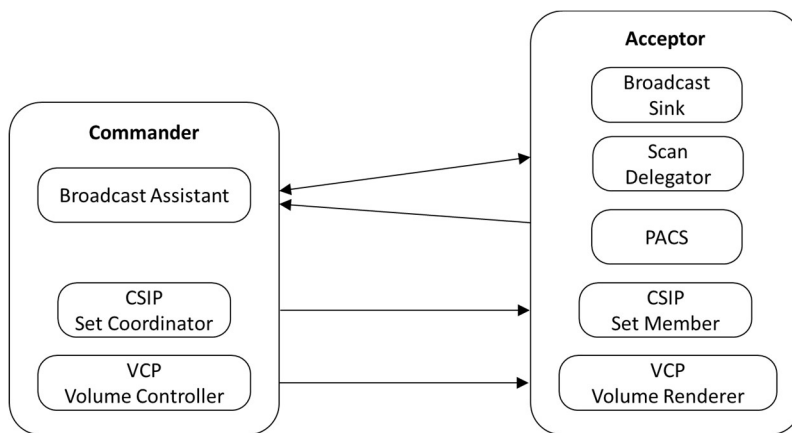


Figure 8.5 The major elements of a Broadcast Assistant (Commander) and a Broadcast Sink (Acceptor)

Figure 8.5 illustrates the major components of a Broadcast Assistant and the main broadcast-

Section 8.6 - BASS – the Broadcast Audio Scan Service

related parts of an Acceptor. Although a Broadcast Assistant is likely to be implemented as a phone app, it can equally be a device in its own right, like a TV remote control. It's useful to visualise it as a separate physical device, because for a user, what it does is very similar to that remote control.

A minimum implementation of a Commander for a Broadcast Sink may just be a remote volume control. In that case it is only acting as a GATT Client, so can be implemented on any Bluetooth LE device. Most implementation will also support the Broadcast Assistant role and scan for Broadcast Sources on behalf of a Broadcast Sink, which requires support for Bluetooth® Core 5.2 or above. We can also implement “half-way” devices which are GATT only, relying on their Broadcast Source to do the scanning. Such a device is described as a virtual Broadcast Assistant, which we'll look at in more detail in Chapter 12.

Commanders for Broadcast Sinks usually contain four main elements:

- A CSIP Set Coordinator, which ensures that any commands are sent to all of the members of a Coordinated Set, whether that is information about a Broadcast Source, or volume control. This ensures that they always operate on all members of a Coordinated Set, which is usually a left and right earbud.
- A Broadcast Assistant, which is used to scan for Broadcast Sources and lets the user select which one they want to listen to,
- A VCP volume controller (which we'll cover in Chapter 10) to control the volume of each member of the Coordinated Set, and
- The ability to obtain and distribute a Broadcast_Code to decrypt broadcast Audio Streams

Without the flexibility that comes with Commanders, and the capabilities they provide, broadcast encryption becomes cumbersome, making use cases around private broadcast streams very difficult to implement. Private broadcasts bring a whole new dimension to Bluetooth LE Audio. Whether that's for personal TV and music, access to a TV in a hotel room, shared audio in a conference room, or the new Bluetooth Auracast™ audio sharing experience, it relies on a simple way for users to connect to the correct Broadcast Source and get access to the Broadcast_Code to decrypt the audio. All of this is managed by two new roles, which are defined in BAP. These are the Scan Delegator and the Broadcast Assistant. They work with the Broadcast Audio Scanning Service (BASS) to distribute the information gathering and control that provide the richness in broadcast applications.

8.6 BASS – the Broadcast Audio Scan Service

First, a few words about BASS - the Broadcast Audio Scan Service. BASS is a service which is instantiated in an Acceptor to expose the details of which broadcast Audio Streams it knows about, which one(s) it's connected to, and whether it wants a Broadcast Assistant to help it manage that information and help it to make connections. BASS works with Broadcast

Assistants (which are defined in BAP, and which we'll get to in a minute) to manage broadcast connections that they select and manage. It's a key part of expanding the broadcast ecosystem, using ACL connections with Acceptors to assist the acquisition and control of broadcast information.

The key elements of BASS are two characteristics, both of which are mandatory whenever it is used:

Characteristic	Properties	Quantity
Broadcast Audio Scan Control Point	Write, Write w/o response	Only one
Broadcast Receive State	Read, Notify	One or more

Table 8.6 BASS characteristics

The Broadcast Audio Scan Control Point characteristic allows one or more Broadcast Assistants to inform an Acceptor of whether they're actively working on its behalf to look for Broadcast Sources. They can tell the Acceptor how to find a BIG, connect to a BIG, disconnect from a BIG and provide the Broadcast_Code to decrypt an encrypted Audio Stream. As we'll see, there is one really important parameter within the Broadcast Audio Scan Control Point characteristic, which is the BIS_Sync. When a Broadcast Assistant sets this to 0b1, it tells the Acceptor to start reception of a stream. When it sets it to 0b0, the Acceptor should stop receiving it.

The "one or more Broadcast Assistants" statement on the first line of the previous paragraph is an important one. An Acceptor can use as many Broadcast Assistants as it likes. All that is required is that each has an ACL link to that Acceptor. Once they're connected and have registered the fact that they're helping the Acceptor to find a broadcast stream, the Broadcast Assistants operate on a first-come, first served basis, limited only by any Locks which are invoked by CSIP if they're operating on a Coordinated Set of Acceptors. Each Broadcast Assistant needs to register with the Broadcast Receive State characteristics on its Acceptor(s) for notifications, which means that all of the Broadcast Assistants will be updated whenever a change is made to the Broadcast Receive State characteristic, either by a Broadcast Assistant writing to it, or as a result of a local action on the Acceptor.

The Broadcast Receive State characteristic instances expose what the Broadcast Sink is doing – which BIG it's connected to, which BISes within that BIG it's currently receiving, whether it can decrypt them and the current metadata it has for each of them. An Acceptor has to have at least one Broadcast Receive State characteristic for each BIG it can simultaneously synchronize to. In Chapter 12 we'll see why it's a good idea to support many more than that.

8.6.1 Broadcast Assistants

Broadcast Assistants allow Acceptors to offload the scanning job that we described in Section 8.3.1. The Broadcast Assistant Role is normally the main Role of a stand-alone Commander,

Section 8.6 - BASS – the Broadcast Audio Scan Service

scanning on behalf of a Broadcast Sink, which is why they're almost always referred to as Broadcast Assistants. There are two important reasons to offload this task. The first is that scanning is a fairly power-hungry operation, which is something that you don't want a hearing aid or earbud to do, or at least not very often, particularly without knowing exactly what it's looking for. It's a job which is far better performed on a device like a phone, or a dedicated remote control, neither of which are being powered by tiny batteries. The second reason is the user experience. Broadcasts should always contain readable metadata in their BASE structures, allowing a device with a display to show what each one is. Displaying that information is something which is easy to implement on a phone or remote control, but largely impossible on an earbud. So, moving the scanning and selection task somewhere else not only saves battery life, it vastly improves the user experience.

Broadcast Assistants have an ACL link with an Acceptor. If the Acceptors are running a CAP based profile, they may be a member of a Coordinated Set, in which case each Commander that includes the Broadcast Assistant has to run the CAP preamble procedure to ensure that it operates on all of the set members. Although Broadcast Assistants are not involved in receiving or transmitting Audio Streams, they can request that Broadcast Sinks synchronize or terminate connections to a broadcast Audio Stream.

Multiple Commanders can be involved, so a user could have a Commander application in a smartwatch as well as their smartphone, and also a dedicated remote control as an additional Commander. Although the Broadcast Assistant is only a role in these devices, they are universally referred to as Broadcast Assistants.

8.6.2 Scan Delegators

Broadcast Assistants work on behalf of a device with a Scan Delegator role. The role is defined in BAP and is normally implemented in a Broadcast Sink. It's job is to find devices taking the Broadcast Assistant Role, which can take over the job of scanning for Broadcast Sources. A Scan Delegator can also be implemented in a Commander, as multiple Commanders can be chained together, effectively relaying information from one to another. We'll see the benefit of that in a later chapter.

The Scan Delegator Role within a Broadcast Sink always contains an instance of BASS. Scan Delegators solicit for Broadcast Assistants which can scan on their behalf by sending solicitation requests using Extended Advertising PDUs which include a Service Data AD Type containing the BASS UUID [BAP 6.5.2].

Any Broadcast Assistant within range can connect and let the Scan Delegator know that it has started scanning on its behalf, interacting through the BASS instance on the Broadcast Sink, where it writes to the Broadcast Audio Scan Control Point characteristic [BASS 3.1], to confirm that it is actively scanning (opcode = 0x01) or has stopped scanning (opcode = 0x00). It is up to the BASS instance to record this status for multiple Clients to determine whether any are actively scanning. This process of scanning on behalf of a Scan Delegator is called

Remote Broadcast Scanning. Once it knows that a Broadcast Assistant is scanning on its behalf, a Broadcast Sink may decide to terminate its own scanning process to conserve power. The Broadcast Assistant can read the PAC records of the Broadcast Sink to discover its capabilities, and use that information to filter the Broadcast Sources it finds, so that it only presents the Scan Delegator with options which are valid for its Broadcast Sink. Most commonly, that would take into account the Broadcast Sink’s Audio Location, Supported and Available Context Types and the Codec Configurations it supports. It can also expose its preferred language.

Once the Broadcast Assistant has detected a suitable Broadcast Source, it can inform the Scan Delegator by writing to its Broadcast Audio Scan Control Point characteristic to start the process of the Broadcast Sink acquiring the broadcast Audio Stream. This could be an automatic action, or it could be user initiated. A user might want to automatically connect to a known Broadcast Source, such as when they walk into a place of worship or their office. Alternatively, they may ask their phone to scan and let them know what’s available using a scanning application, before selecting their preferred Broadcast Source. That choice is implementation specific.

8.6.3 Adding a Broadcast Source to BASS

Once that choice has been made by the user, the Broadcast Assistant writes it to the first empty Broadcast Receive State characteristic on the Broadcast Sink, adding the selected Broadcast Source information [BAP 6.5.4]. Before it does this, it should have read the current status of the Broadcast Receive State characteristics. If the user has chosen a Broadcast Source which already exists in one of them, then it should update the appropriate Broadcast Receive State characteristic using the Modify Broadcast Source procedure [BAP 6.5.5]. It should also check that the Broadcast Sink is capable of decoding any proposed BIS.

The parameters that it writes into the characteristic using the Add Source Operation opcode of 0x02 [BASS 3.1.1.4], fall into three broad categories, as shown in Table 8.7.

Parameter		Size (octets)	Description
<i>Source Information</i>			
	Advertiser_Address_Type	1	Public (0x00) / Random (0x01)
	Advertiser_Address	6	Advertising address of the Broadcast Source
	Advertising_SID	1	ID of the advertising set
	Broadcast_ID	3	Broadcast ID of the Broadcast Source
<i>Action</i>			
	PA_Sync	1	0x00: Don’t synchronize

Parameter		Size (octets)	Description
			0x01: Synchronize using PAST 0x02: Synchronize without using PAST
	PA_Interval	2	The SyncInfo field Interval
	Num_Subgroups	1	Number of subgroups in the BIG
	BIS_Sync[i]	4	BIS_Index values to connect to: 0x00: Don't synchronize to BIS_Index[i[k]] 0x01: Synchronize to BIS_Index[i[k]]
Configuration & Content			
	Metadata Length[i]	1	Metadata length for the [i th] subgroup in the BIG
	Metadata [i]	Varies	Metadata for the [i th] subgroup in the BIG

Table 8.7 Format of the Add Source Operation

The first set of parameters – the Source Information, informs the Broadcast Sink of the identity of the Broadcast Source and the BIG, so that it knows which device (or more correctly, device identity) this information refers to. The Advertising Set ID – the SID, which is contained in the primary advertisements and is defined when the Extended Advertising is set up [Vol 4, Part E, Section 7.8.53], generally stays static for the life of a device, and isn't allowed to change between power cycles. Although it is only a single octet, it can help to identify the device. The Broadcast_ID is in the AUX_ADV_IND of the Extended Advertising and is static for the lifetime of a BIG (which may be shorter than the SID). Along with the advertising addresses, this is enough information for a Broadcast Sink to scan for the Broadcast Source.

The next group of parameters tells the Broadcast Sink what it should do. This would normally be triggered by the user selecting a Broadcast Source on the user interface of their Commander. PA_Sync tells it to synchronize to the periodic advertising train, which lets it obtain the BASE and BIGInfo. Setting the PA_Sync to either 0x01 or 0x02 tells the Broadcast Sink to go and do that. The PA_Interval is the interval between successive AUX_ADV_IND packets and if known, can help the scanner. Num_Subgroups provides the number of subgroups in the BASE, followed by the most important parameter, which is the BIS_Sync.

BIS_Sync is the instruction from the Broadcast Assistant to the Broadcast Sink to tell it which specific broadcast stream or streams to connect to. It's a four octet wide bitmap of all of the BISes, with values set to 0x01 for any stream which the Acceptor should connect to. There's a slight subtlety here, which is why it's arrayed, rather than a single octet, which is that it is effectively masked for each subgroup. So, each BIS_Sync[i] is only valid for that subgroup –

all other values are set to 0b0. The reason for that is that an Acceptor is never expected to receive BISes from different subgroups at the same time in the same device, or set of devices, as the whole point of a subgroup is to collect associated streams which should be rendered together. If two subgroups carried different languages, say Japanese and German, the expectation is that you would only choose one.

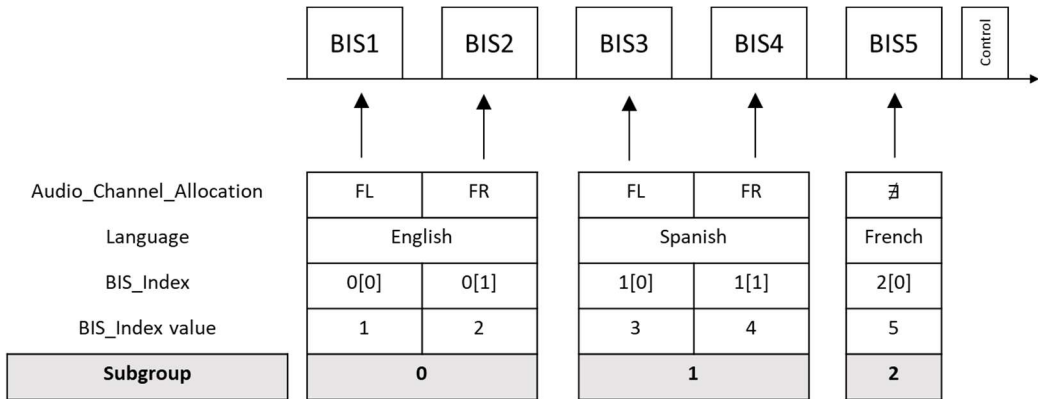


Figure 8.6 Illustration of mapping from BIS number to BIS_Index (∅ indicates it does not exist)

Figure 8.6 illustrates how BIS_Index values in subgroups relate to the ordering in a BIG, where BISes are arranged in numeric order. In this case, the BIS_Index values align with the BIS numbering, but they don't need to. Implementations should be flexible to cope with a mismatch. BIS 5 does not contain an Audio_Channel_Allocation LTV, indicating that the French stream is mono.

In writing a BIS_Sync value, only one subgroup can be selected. So, if subgroup 0 were selected from Figure 8.6, the only allowable values for BIS_Sync[0] would be for one or both of bits 0 and 1 set to 0b1.

A Broadcast Assistant can set the value of BIS_Sync to 0xFFFFFFFF, meaning “no preference”, in which case the Broadcast Sink is free to make its own decision regarding which BISes to use. A Broadcast Sink can always decide to override the value that is written to its Broadcast Audio Scan Control Point characteristic.

Finally, the metadata fields provide the Level 2 metadata for each subgroup – typically the Audio Contexts and the Audio Locations, allowing the Acceptor to decide whether they are appropriate for what it wants to do, i.e., do they match its current Available Bluetooth LE Audio Context Type settings. The data sent in the Add Source operation, which is written into each Broadcast receive state can be modified at any time by the Modify Source operation, which will update the current information. Despite the specific synchronization information which is conveyed in these two operations, it is entirely up to the Acceptor whether it follows the request to synchronize with the PA and the BISes which are written to it. That's up to the implementation.

Section 8.6 - BASS – the Broadcast Audio Scan Service

If the Acceptor decides to act on the instruction from the Broadcast Assistant, it will use the instructions to synchronize to the Periodic Advertising train, either by using PAST, or scanning with the Broadcast Source information it's been given, retrieving the BASE from the Basic Audio Announcement Service data and the BIGInfo from the ACAD in the Periodic Advertising Train (AUX_SYNC_IND). These give the Acceptor's controller all of the information it needs to synchronize to the BIG, after which it will use the BIS_Sync value to select which specific BIS or BISes it receives.

PAST⁶⁵ – the Periodic Advertising Synchronization Transfer procedure [Core, Vol 3, Part C, Section 9.5.4] is the most efficient method for a Broadcast Sink to use for synchronization, as the Broadcast Assistant provides the Scan Delegator with the SyncInfo data to use in conjunction with a timing reference anchor from the ACL link of the Commander, allowing it to synchronize directly to the PA. This process is known as Scan Offloading.

Disconnection can be performed autonomously by the Acceptor, or a Broadcast Assistant can use the Modify Source operation to set the appropriate BIS_Sync bit to 0b0. It may also tell the Acceptor to stop synchronizing to the PA, but retain the rest of the Source information. Alternatively, it can delete the Source record, by performing the Remove Source operation on that Source_ID.

8.6.3.1 Set, Source and Broadcast IDs

It's worth a quick précis of the different IDs involved in these operations, as they can be confusing.

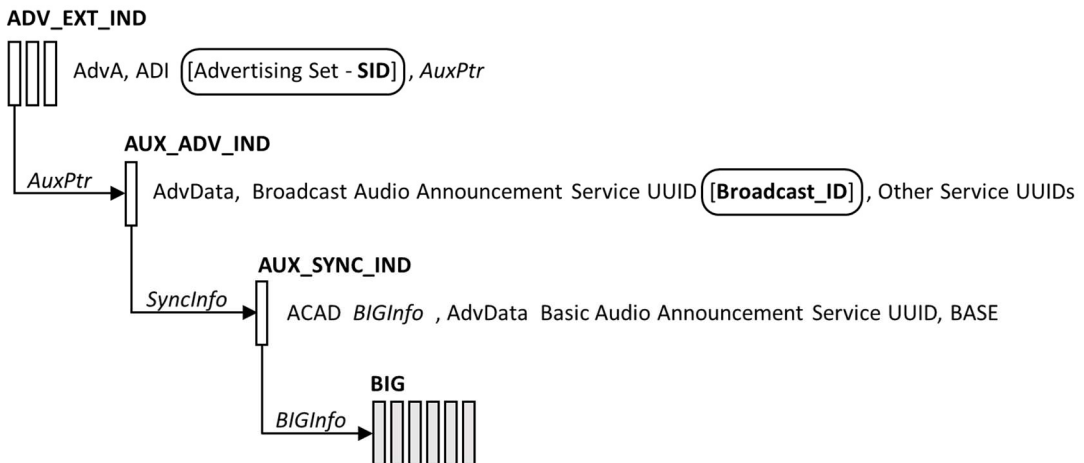


Figure 8.7 Set and Broadcast IDs

⁶⁵ The acronym PAST does not exist in the Core, but has been introduced in BAP. It does not affect the Core definition.

Referring to Figure 8.7, the Set and Broadcast IDs are properties of the Broadcast Source. The Advertising Set ID is included in the primary advertisements and identifies a specific set of Extended and Periodic Advertising trains. The SID is normally static for the lifetime of the device, but must remain unchanged within every power cycle. The SID can be useful when an Acceptor or Commander wants to reconnect regularly to a known source, whether that's a personal device, or a fixed, infrastructure Broadcast Transmitter. It should be different for each BIG transmitted by a Broadcast Source. It is a 4 bit number set by the Initiator and is defined in the Core Vol 6, Part B, Section 2.3.4.4.

The Broadcast_ID is specific to a BIG and is contained in the Broadcast Audio Announcement Service UUID. It remains static for the lifetime of the BIG. It is a 3 octet random number, defined in BAP 3.7.2.1.1, which is normally static for the life of the device.

The Source_ID, which we'll look at in more detail in the next section, is generated independently by each Broadcast Sink and is used to identify a specific set of broadcast device and BIG information. It is local to an Acceptor and used as a reference for a Broadcast Assistant. In the case of a Coordinated Set of Acceptors, such as a left and right earbud, the Source_IDs are not related and may be different, even if both are receiving the same BIS, as each Acceptor independently creates their own Source ID values. A Broadcast Assistant which is working with a Coordinated Set of devices will use the Coordinated Set SIRQ value to correlate the Source_IDs of entries in their Broadcast Receive State characteristics. The Source_ID is defined in Section 3.2 of BASS.

8.6.4 The Broadcast Receive State characteristic

On an Acceptor, the Broadcast Receive State characteristic is used to inform any Broadcast Assistant of its current status regarding broadcast Audio Streams. There are typically multiple instances of this characteristic on each Acceptor. Its structure is shown in Table 8.8.

Field	Octets	Description
Source_ID	1	Assigned by the Acceptor
Source_Address_Type	1	From the Add / Modify Source operation or an autonomous action
Source_Address	6	From the Add / Modify Source operation or an autonomous action
Source_Adv_SID	1	From the Add / Modify Source operation or an autonomous action
Broadcast_ID	3	From the Broadcast Audio Announcement Service UUID

Section 8.6 - BASS – the Broadcast Audio Scan Service

Field	Octets	Description
PA_Sync_State	1	Current synchronization status to the PA: 0x00 – Not synchronized to the PA 0x01 – SyncInfo request 0x02 – Synchronized to the PA 0x03 – Synchronization failed 0x04 – No PAST Other values – RFU
BIG_Encryption	1	Encryption status: 0x00 – Not encrypted 0x01 – Broadcast_Code required 0x02 – Decrypting 0x03 – Bad_Code (wrong encryption key)
Bad_Code	Varies	Not present unless BIG_Encryption = 0x03 (wrong key), in which case it contains the value 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF. ⁶⁶
Num_Subgroups	1	Number of subgroups
BIS_Sync_State[i]	4	BIS synchronization state for the i th subgroup
Metadata_Length[i]	1	Length of the metadata for the i th subgroup
Metadata[i]	varies	Metadata for the i th subgroup

Table 8.8 Format of the Broadcast Receive State characteristic

As with other characteristics which can be set by Client operations, as well as being changed by an autonomous Server action, the Broadcast Receive State characteristic can be used by a Client to check the Acceptor’s status, and also be notified by the Server to denote that an operation is complete, such as synchronizing to a PA, or to request an action from a Client at the Link Layer level. When they’re being notified, some of these act as error reports or requests for action. Amongst these, the most important are notifying a PA_Sync State of 0x01 to request SyncInfo and notifying a BIG_Encryption state of 0x03 to request a Broadcast_Code.

Each Acceptor will normally have multiple instances of the Broadcast Receive State characteristic, allowing it to hold information on multiple Broadcast Transmitters within range. These can be populated by one or more Broadcast Assistants using the Add Source or Modify Source commands, or independently by the Acceptor. Any autonomous change of a Broadcast Receive State characteristic, either by scanning for new Broadcast Transmitters, a change of BIS synchronization as a result of a local user actions, or the loss of a BIS should

⁶⁶ In earlier versions of BASS, this field contained the value of the “bad” 16 octet Broadcast_Code. This was considered to be poor practice, as it exposed what may have been a previously valid code. To prevent this, devices must now return the value 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF.

be notified to all of the device's Broadcast Assistants notifying the change to the Broadcast Receive State characteristic instance.

8.7 Broadcast_Codes and encrypted broadcasts

The ability to encrypt broadcast Audio Streams takes Bluetooth LE Audio into to a whole new area of audio applications. Encryption effectively provides a pseudo-geofencing capability, limiting broadcast coverage to those who have a decryption key. That can be used to limit access to overlapping broadcasts in a specific area, in the same way that Wi-Fi codes are used. For example, a hotel could allocate codes to prevent people listening to audio from an adjoining meeting room, or a TV on the floor above. By providing the means to send the encryption key over a Bluetooth link, or to allow a Broadcast Assistant to obtain it by using an out of band method, it becomes even more powerful. In Chapter 11 and 12, we'll investigate the different ways that the Broadcast_Code can be distributed for different applications.

Personal devices which implement audio sharing, such as TVs, phones, tablets and laptops, can collocate a Broadcast Assistant with the Broadcast Source. The Broadcast Assistant has direct access to the Broadcast_Codes and can write them directly to the Broadcast Sink. It requires the Broadcast Sinks to be paired with the Broadcast Source, but this would be expected for personal devices.

The life of a Broadcast_Code is implementation dependent. In some case it could be permanent – a Broadcast Source playing music in a coffee-shop would probably never change its code. A TV in a hotel room would maintain its Broadcast_Code for as long as the guest was staying in that room; a personal TV might renew it on each power cycle, so that friends who visited didn't keep it indefinitely, whilst a mobile phone app which shared audio with your friends would probably renew it every session. These different lifetimes mean that Broadcast Sources and Broadcast Assistants need to support a variety of different methods to acquire the respective Broadcast_Codes.

8.8 Using Broadcast Assistants to select broadcast Audio Streams

We can now revisit how devices are likely to receive Broadcast Streams in the real world, which will almost always involve a Broadcast Assistant, whether that's a stand-alone device, a wearable, an app on a phone, or a Broadcast Assistant built into the Broadcast TV.

8.8.1 Solicitation Requests

If a Broadcast Sink wants to use a Broadcast Assistant to find Broadcast Sources, it uses its Scan Delegator role to send out Extended Advertisements containing a Service AD Data Type which includes a Broadcast Audio Scan Service UUID. These are called Solicitation Requests [BAP 6.5.2].

Section 8.8 - Using Broadcast Assistants to select broadcast Audio Streams

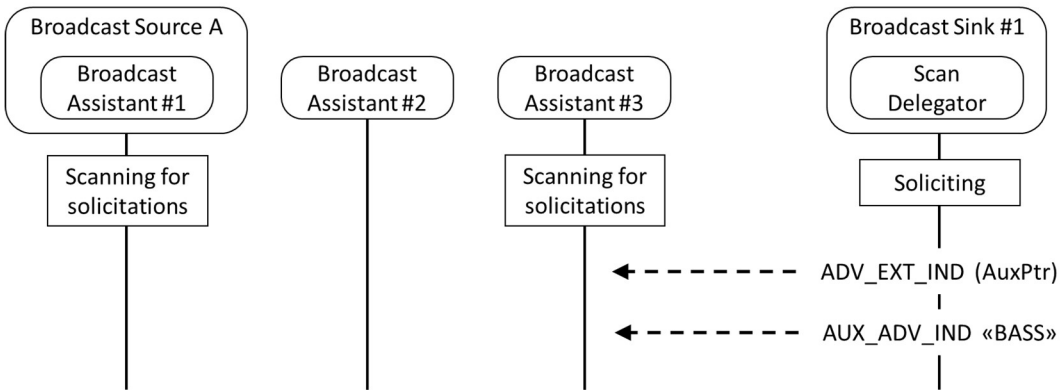


Figure 8.8 The Scan Delegator solicitation process

Figure 8.8 shows a Scan Delegator on a single device sending out Extended Advertisements containing the BASS Service UUID. To the left, three previously paired Broadcast Assistants are within range. Broadcast Assistant #1 is collocated with a Broadcast Source and is actively scanning. Broadcast Assistants #2 and #3 are devices with just the Commander role, but only Broadcast Assistant #3 is actively scanning. In this case, both Broadcast Assistants #1 and #3 could respond to the Solicitation requests.

In many cases, there may be a pair of earbuds. CAP, as always, tells each Broadcast Assistant to start off by connecting, establishing the members of any Coordinated Set it finds, then lays out the BAP procedures to follow. At this point, each Broadcast Assistant should read each Broadcast Sink’s PAC records to determine their capabilities, mainly because there’s no point in them telling a Broadcast Sink about broadcast Audio Streams that the Broadcast Sink can’t support. There may be many reasons for making that decision. It could be because the use case of the broadcast Audio Stream has a Context Type the Broadcast Sink can’t support; it might have a codec configuration it can’t decode, have an incompatible Channel Allocation or Audio Location, can’t support encryption, etc.

Only one Acceptor of a Coordinated Set needs to perform the Solicitation process. In most cases, there is a practical reason for this, which is that it is unlikely that both would have room for a dedicated button on their limited user interface. Once the Broadcast Assistant responds and determines that it is a member of a Coordinated Set, CAP requires it to find the other members, and from that point, read all of their PAC records, then interact with the instances of BASS on each of them.

The Broadcast Assistant needs the PAC information to be able to set the appropriate BIS_Sync value in each device, to ensure that it receives the correct BIS – typically by specifying the left or right streams each Acceptor should synchronize to. However, a Broadcast Sink may overrule that decision, if it has set local rules based on metadata which is not exposed in PACS, such as a preferred language.

Figure 8.8 shows a simplified example of that process for Broadcast Assistant #3, which is responding to a Solicitation request by one member of a Coordinated Set. It connects to the Scan Delegator in Broadcast Sink #1, discovers that it is a member of a Coordinated Set of two Acceptors, then finds and connects to the second member.

It discovers and reads the PAC records of each Acceptor and will set its broadcast filter policy so that it only reports broadcast Audio Streams which can be received by both Acceptors.

Next, it will discover and read the BASS Broadcast Receive State characteristics and set up notifications for these characteristics, so that it will be aware of any changes. Reading the states tells it whether the Acceptors are currently synchronized to any Periodic Advertising train or are receiving any BISes. Once it has completed these processes, it is ready to start scanning on behalf of the Acceptors, and writes to their Broadcast Audio Scan Control Point characteristic to inform them of this.

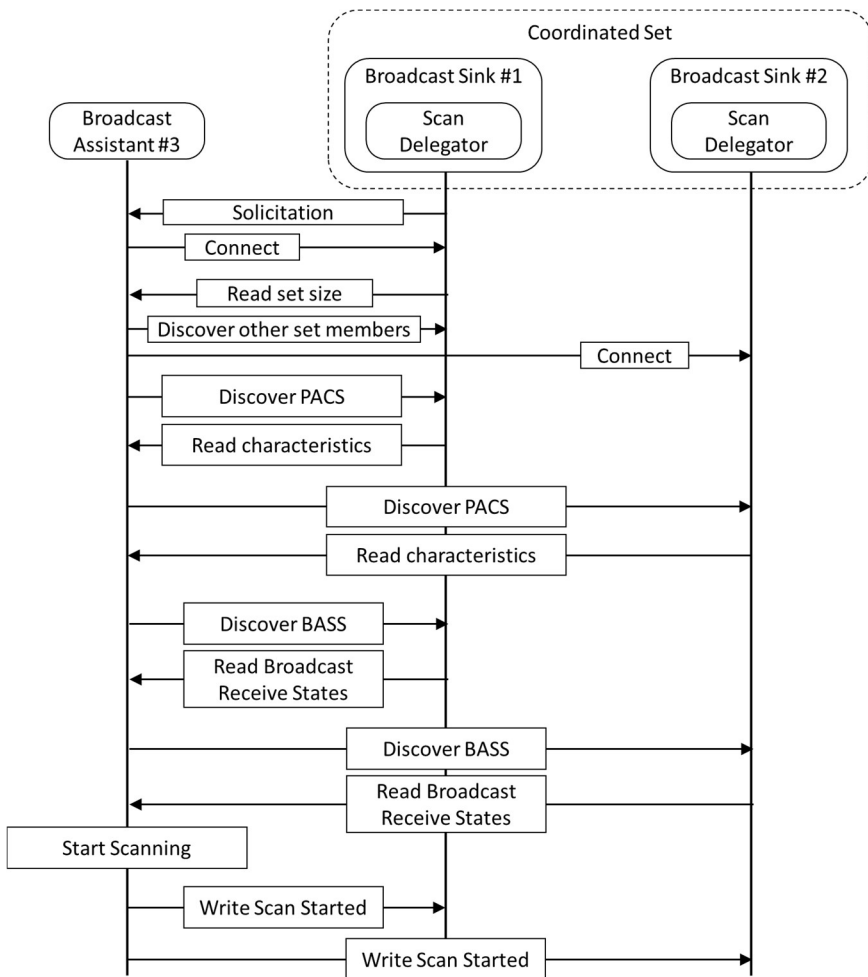


Figure 8.9 Discovery of Broadcast Sink Capabilities and State by a Broadcast Assistant

Section 8.8 - Using Broadcast Assistants to select broadcast Audio Streams

Once that's done and there is at least one active Broadcast Assistant scanning on behalf of the Scan Delegator, we move into BAP's Broadcast Assistant procedures, starting with Remote Broadcast Scanning [BAP 6.5.3].

8.8.2 Remote Broadcast Scanning

A Broadcast Assistant that has informed the Scan Delegator that it is scanning will start searching for Broadcast Transmitters. As it finds each Advertising Set, it will work its way through the Extended and Periodic Advertisements (AUX_ADV_IND and AUX_SYNC_IND), examining the contents and parsing the structures to determine whether the broadcast Audio Streams match the capabilities of the Acceptor. In most cases, it will build up a list of relevant, available broadcast Audio Streams and present them to the user. If the Commander has a suitable user interface, such as an app on a smartphone, this will generally be presented in the form of a sorted list, from which the user can select a broadcast Audio Stream to connect to.

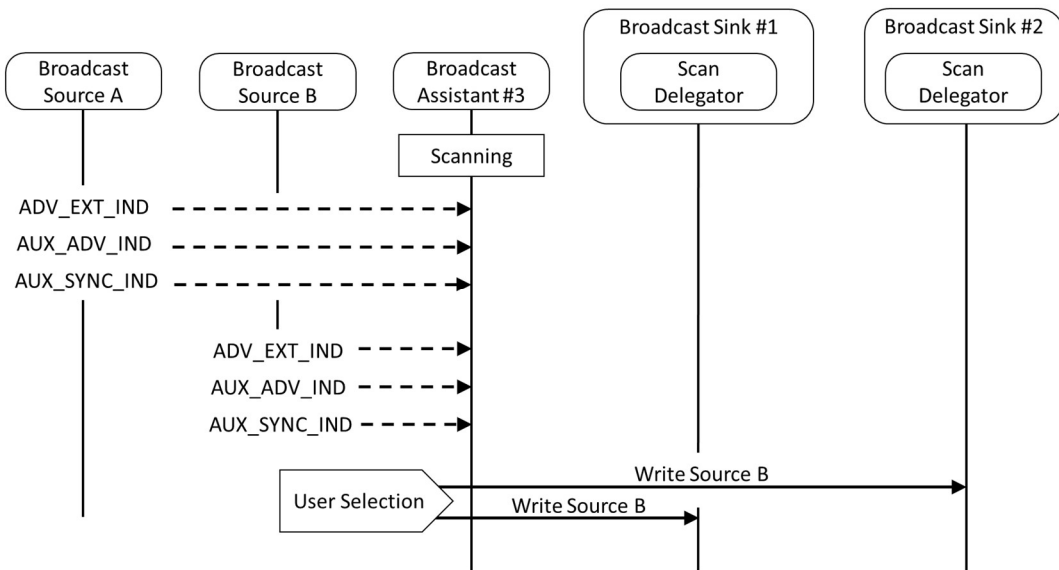


Figure 8.10 Remote broadcast scanning

Figure 8.10 follows on from Figure 8.9, where Broadcast Assistant #3 is scanning on behalf of the two Broadcast Sinks. It has discovered Broadcast Source A and Broadcast Source B. The user makes a decision on the Broadcast Assistant to listen to Broadcast Source B and selects it, at which point Broadcast Assistant #3 writes an Add Source or Modify Source command to the Broadcast Audio Scan Control Point characteristics in the Scan Delegators of the two Broadcast Sinks, requesting them to synchronize with Broadcast Source B.

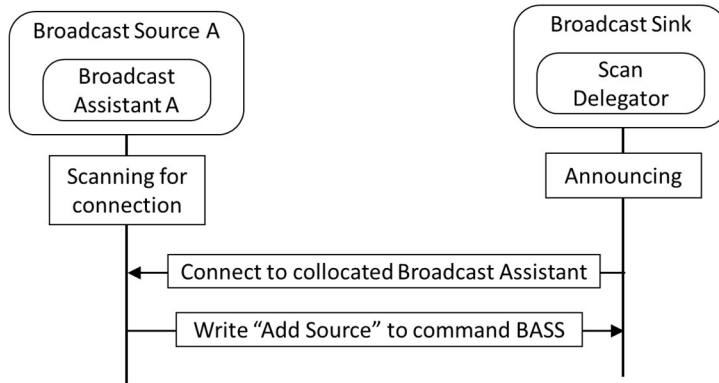


Figure 8.11 Collocating a Broadcast Assistant with a Broadcast Source

Figure 8.11 shows how a collocated Broadcast Assistant behaves. In this case, Broadcast Assistant A is collocated with Broadcast Source A. Broadcast Assistant A has never written to the Broadcast Scan Control Point to say it is scanning on behalf of the Scan Delegator. Instead, its purpose is to provide the Scan Delegator with information about its collocated Broadcast Source. On receiving a connection request from a previously paired Broadcast Sink, it writes an Add Source or Modify Source operation to the Broadcast Sink’s Broadcast Audio Scan Control Point characteristic, instructing it to synchronize to its Broadcast Source’s audio streams.

A further advantage of collocation is that the Broadcast Assistant is aware of the Broadcast_Code for encrypted transmissions. This allows it to provide it to the Broadcast Source using the Set_Broadcast_Code operation on the Broadcast Audio Scan Control Point characteristic, within any additional user interaction. This is particularly useful for applications where the Broadcast_Code is changed on a session basis, as is likely with personal or hotel room TVs.

A collocated Broadcast Assistant can also perform scanning, in which case it would have told the Scan Delegator that it is scanning on its behalf, but that is an implementation choice. Note that Broadcast Assistants should always update their local information from notifications sent from their Broadcast Sink(s).

A practical point to bear in mind is that the receivers in Broadcast Assistants may have better sensitivity than the receivers in earbuds, as they are likely to have larger antennas. This means that they will probably detect Broadcast Sources which would be out of range of their Broadcast Sinks. Implementations may want to determine the RSSI of signals from Broadcast Sources and use that information to arrange the order of presentation of Broadcast Sources they have found to the user. However, a collocated Broadcast Assistant may never scan, but confine itself to only providing information about its collocated Broadcast Source.

Section 8.8 - Using Broadcast Assistants to select broadcast Audio Streams

8.8.3 Receiving a Broadcast Stream

We are now at the point where the user has decided what they want to listen to, which takes us back to the operation of Adding or Modifying a Broadcast Source, which we described in Section 8.6.3.

The process of synchronizing to a BIS normally involves a Client action on the Broadcast Audio Scan Control Point characteristic. This may be local, but is usually the result of an event triggering an operation by a Broadcast Assistant. The resulting Server behaviour is exposed by one of the Broadcast Sink's Broadcast Receive State characteristics. Table 8.9 reminds us of the relevant fields in the two server characteristics, and Figure 8.12 shows how they interact with each other.

PA_Sync Field (Broadcast Audio Scan Control Point)	
0x00	Do not Synchronize to PA
0x01	Synchronize to PA – PAST available
0x02	Synchronize to PA – PAST not available

PA_Sync_State Field (Broadcast Receive State)	
0x00	Not Synchronized to PA
0x01	Syncinfo Request
0x02	Synchronized to PA
0x03	Failed to Synchronize to PA
0x04	No PAST

Table 8.9 Synchronization fields of the Broadcast Audio Scan Control Point and Broadcast Receive State characteristics

The operation on the Broadcast Audio Scan Control Point characteristic can initiate synchronization to a PA (using the PA_Sync field), synchronization to specific BISes (using the BIS_Sync[i]) field, or both. However, this is a two stage process in the Broadcast Receiver. It needs to synchronize to the PA first to obtain the BASE and BIGInfo before it can synchronize to a BIS. A Broadcast Assistant can send two separate commands, first requesting synchronization to the PA, then, when notified that this is complete, send a Modify Source command requesting synchronization to the BIS. Broadcast Receivers should be able to cope with both stages.

In most cases a Broadcast Receiver will remain synchronized with both the PA and a BIS, but it does not need to be. It can drop synchronization to the PA after it has synchronized to a BIS, but this will prevent it from seeing any changes in BASE metadata.

Although a Broadcast Receiver can scan and synchronize autonomously, in most cases synchronization to a broadcast will occur as the result of a remote client action, with a Broadcast Assistant writing to the Add Source or Modify Source fields on the Acceptor's Broadcast Audio Scan Control Point characteristic.

If the Client writes a value of 0x00 to the PA_Sync field, then no further action occurs – it is simply updating the Acceptor’s database of available Broadcast Transmitters.

If the Client writes a value of 0x01 or 0x02 to the PA_Sync field, then the Acceptor will attempt to synchronize to the Periodic Advertising, using the process shown in Figure 8.12. This involves it acquiring the SyncInfo, after which it can synchronize to the PA, obtains the BIGInfo and BASE, and synchronize to one or more BISEs.

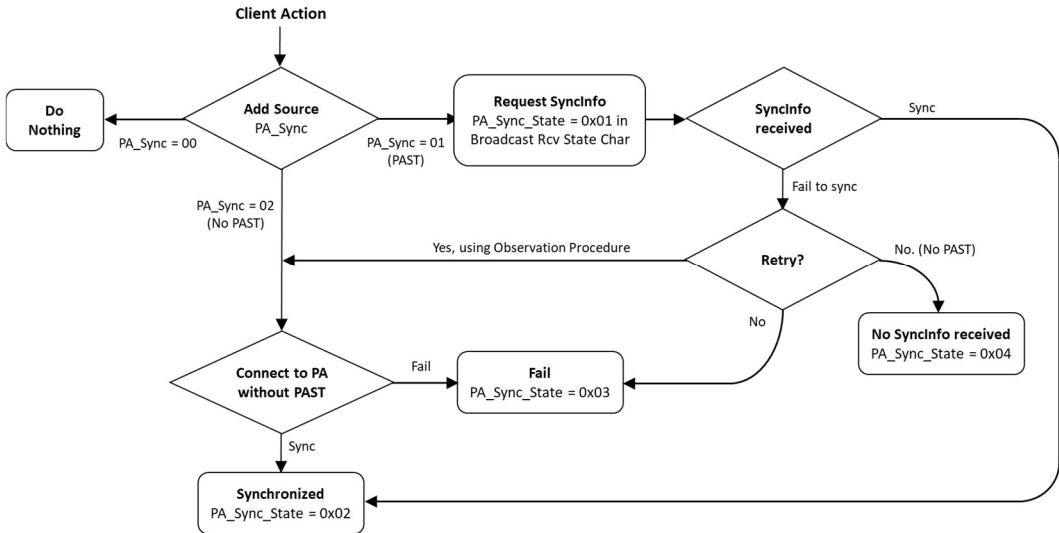


Figure 8.12 The process of a Broadcast Receiver synchronizing to a Periodic Advertising train

In Figure 8.12, illustrates the two BASS characteristics involved. To start the process, the PA_Sync field in the Broadcast Audio Scan Control Point characteristic is written to instigate an action on the Broadcast Receiver. There are three options, which write the following opcodes based on the desired action:

- 0x00 - Do not synchronise to the PA. This generates (Add Source) or updates (Modify Source) an instance of the Broadcast Receive State characteristic, but does not request synchronization to the PA. It is predominantly used to update the Broadcast Receiver’s database of available Broadcast Sources. It may also be used to terminate a current synchronization.
- 0x01 – A request to synchronize to the Periodic Advertising train using PAST. The Client must be able to support a request to supply the SyncInfo if it writes this opcode.
- 0x02 – A request to synchronize to the Periodic Advertising train without using PAST to obtain the SyncInfo. This requires that the Broadcast Receiver uses the Observation procedure from the Core specification, followed by the Periodic Advertising Synchronization Establishment procedure to obtain the SyncInfo. It

Section 8.8 - Using Broadcast Assistants to select broadcast Audio Streams

requires the Broadcast Receiver to perform its own scanning, which will have a battery impact, so should be avoided by Broadcast Assistants which can support PAST.

The Broadcast Receive State characteristics on the Broadcast Receiver contain a complementary PA_Sync_State field. The four values of this field are status codes that represent the state of synchronization for each Broadcast Receive State characteristic instance:

- 0x00 – not synchronised this Broadcast Source's PA
- 0x02 – synchronized to this Broadcast Source's PA
- 0x03 – failed to synchronize to this Broadcast Source's PA
- 0x04 – failed to obtain a SyncInfo (no PAST)

All of these are notified whenever the value changes.

The 0x01 value is interesting, as it is a request to a client to provide it with the SyncInfo using PAST. It should only be set if the Server supports PAST, and has received a Client opcode of 0x01 written to its Broadcast Audio Scan Control Point characteristic.

The values of the PA_Sync_State are notified, so it is possible that multiple Broadcast Assistants will see the request and respond. Designers may want to consider this and only respond when a Broadcast Assistant has previously sent a command and was expecting the notification.

If the Broadcast Assistant fails to synchronise using PAST, or if it does not receive the SyncInfo, it can abort the attempt, writing 0x03 (Failed to Synchronize), or 0x04 (didn't receive a SyncInfo) to the PA_Sync_State field of its Broadcast Receive State Characteristic. Alternatively, it can revert to attempting to connect using the Observation Procedure [Core Vol 3, Part C, Section 9.1.2] to find the Extended Advertising and extract the SyncInfo data. This is the same procedure that it would have used if it had been informed that PAST was not available (i.e. the Client would have written 0x02 to the Broadcast Audio Scan Control Point characteristic). Performing the Observation procedure is a slower process than PAST, requiring the Acceptor to do more work, but it is supported by all Broadcast Transmitters and Receivers. At the end of the procedure, the Broadcast Receiver should have retrieved the SyncInfo, so can proceed to synchronise to the Periodic Advertising train by using the Core's Periodic Advertising Synchronization Establishment procedure [Core Vol 3, Part C, 9.5.3]. Once it is synchronized, it should obtain the BIGInfo and BASE in the same way that it would if it had supported PAST.

If successful, the Acceptor would write the value of 0x02 to the PA_Sync_State of the appropriate Broadcast Receive State characteristic.

In most cases, when the user selects a Broadcast Source on their Broadcast Assistant, it will not just set the PA_Sync, but also the BIS_Sync parameters to instruct each Acceptor which BIS it should select. This will be based on what the Broadcast Assistant has learnt from reading their PACS records. However, Acceptors should be able to make their own selection, as the Broadcast Assistant has an option to leave the choice open.

As multiple Acceptors in a Coordinated Set act independently, they may take different amounts of time to obtain the PA train before they can synchronize with the appropriate BISes, particularly if they have to scan rather than using PAST. That could result in a noticeable delay between the time that audio starts being rendered in two earbuds. The Bluetooth specifications do not contain any details on how to address this, leaving it to implementation. If designers add a proprietary link between left and right earbuds or hearing aids, they should consider using this to support a clean start to rendering.

8.8.4 Broadcast_Codes (revisited)

If we combine our previous diagrams into Figure 8.13, we have two Broadcast Sources (A and B), with Broadcast Source A colocated with a Broadcast Assistant #1. If we assume that the broadcast Audio Streams from both Broadcast Source A and Broadcast Source B are encrypted, there's likely to be a slightly different way in which the two Broadcast_Codes would be obtained for Broadcast Sink #1.

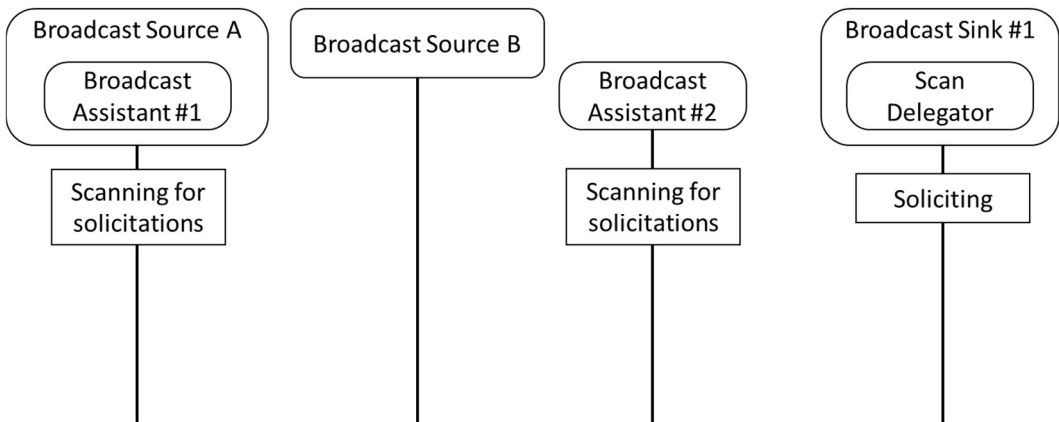


Figure 8.13 Obtaining Broadcast_Codes

Broadcast Source A's Broadcast Assistant knows the Broadcast_Code for its streams – that's why it's colocated. Broadcast Sink #1 has an ACL link to Broadcast Assistant #1, so as soon as the user selects Broadcast Source A, the Broadcast_Code will be sent over in the Set Broadcast Code Command. No further user interaction is required.

In the case of Broadcast Source B, neither Broadcast Assistant #1 nor #2 may know the Broadcast_Code. In this case, Broadcast Sink #1 can ask all of its Broadcast Assistants if they know the Broadcast_Code, by notifying the appropriate Receive State characteristic with the

Section 8.8 - Using Broadcast Assistants to select broadcast Audio Streams

BIG_Encryption field set to 0x01, indicating that it requires the Broadcast_Code. If the Broadcast Sink already has a Broadcast_Code for this stream, which no longer works (normally because it was obtained in an earlier session), it should set the BIG_Encryption field to 0x03 to indicate an incorrect Broadcast_Code. Any Broadcast Assistant can respond to this notification if it knows the code, by writing the Broadcast_Code value to the Scan Delegator's Broadcast Audio Scan Control Point using the Set Broadcast Code operation (0x04) [BASS 3.1.1.6], along with the Source_ID.

If none of the Broadcast Assistants have the correct code, then Broadcast Sink #1 or one of the Broadcast Assistants will have to revert to an out of band method to obtain it. In Chapter 12 we will see just how important Broadcast_Codes are for new audio applications and look at a range of out-of-band methods by which a Broadcast Assistant may obtain a Broadcast_Code, opening up some interesting new use cases.

8.8.5 Proprietary links between Broadcast Sinks and notifications

Almost every pair of earbuds and hearing aids that have ever been shipped include a proprietary link radio between the left and right units. Generally, it's a sub-GHz radio or a Near Field Magnetic Induction (NFMI) link. A few large hearable devices which have room for extended antennas may use Bluetooth, but as 2.4GHz is strongly absorbed by the body, it rarely works for ear-to-ear communication.

The reason for these links is that the controls for independent ear-worn devices are generally not duplicated, and an action on one device uses the proprietary link to reflect it on the other device. When a user presses a button to connect to a telecoil loop, or to change the volume, both devices react as a pair.

The Bluetooth LE Audio specifications do not define any direct connection between two different Acceptors. Instead, they rely on Commanders, whether they are stand-alone devices or roles colocated with an Initiator to ensure that both Acceptors receive the same control information. That task falls to Initiators for unicast streams and volume controllers for volume settings. With Broadcast applications, if there is no proprietary connection ensuring that both Acceptors operate as a single entity, the operation of a Coordinated Set relies on the presence of a Commander which reacts to notifications.

8.8.5.1 The importance of notifications

As far as the Bluetooth LE Audio specifications are concerned, members of a Coordinated Set act independently of each other⁶⁷. Each holds the state for its volume, volume offset, audio input, microphone input, and Broadcast Receive State information in the appropriate characteristics. These values can be changed by one or more devices acting as the Client, or

⁶⁷ There is one exception, which is in the Hearing Access Profile and Service, where a hearing aid can inform a Hearing Aid Remote Controller that actions on a Preset will be locally synchronized with the other hearing aid.

be adjusted locally through a user interface. Every Client device can, and should register for notifications, which will provide an update whenever the value of the state of any member of that Coordinated Set changes. The Client should then write the appropriate value to the other member(s) of that Coordinated Set.

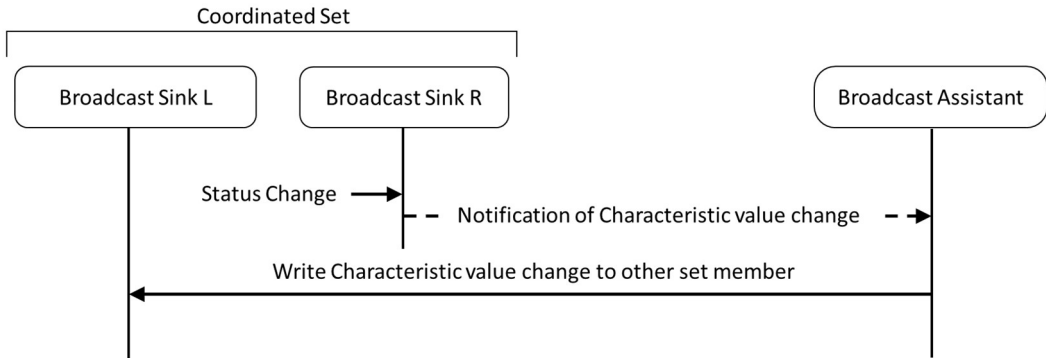


Figure 8.14 Use of notifications to synchronize state information

Figure 8.14 illustrates how this works. In this example, a status change has been made locally on the right Broadcast Sink. This is notified to the Broadcast Assistant, which is acting as a Client for that characteristic. Having received the notification, the Broadcast Assistant should write the appropriate value to the left Broadcast Sink to ensure they remain synchronized. Depending on the parameter being notified, this may not be the same value for each earbud. In the case of synchronizing to a Broadcast Source, it should be the same subgroup of audio streams from the BASE, but one earbud is likely to be directed to the Front Left stream, and the other to the Front Right. For some volume operations, the Client may need to retrieve the change counter value before applying the newly notified value.

The resulting change will be notified. Developers should be aware of this stream of consequential notifications. Acceptors should not notify a write from a Client if it does not result in a change to the current value.

8.8.6 Ending reception of a broadcast Audio Stream

Termination of a broadcast Audio Stream may be performed autonomously by an Acceptor, in which case it will notify this change in its Broadcast Receive State characteristic. If a Commander receives this notification and is aware that the Acceptor is a member of a Coordinated Set, it may decide to terminate the reception on the other Acceptor(s) by writing the appropriate value in their Broadcast Audio Scan Control Points. However, this is implementation specific.

Alternatively, a user can use a Commander to stop reception by writing to the Broadcast Audio Scan Control Points of each Acceptor, with the BIS_Sync value set to zero for all BISes in all subgroups and the PA_Sync value set to 0x00. Once the Acceptor indicates that it is no longer synchronized to either a BIS or a PA by notifying its Broadcast Receive State characteristic with both PA_Sync_State and BIS_Sync_State[i] set to zero, the Broadcast Assistant may

Section 8.9 - Handovers between Broadcast and Unicast

perform the Remove Source Operation (0x05) [BASS 3.1.1.7] to remove the associated Broadcast Receive State characteristic. However, it may leave this to the discretion of the Broadcast Sink. If the Broadcast Sink has dropped the broadcast stream to take a phone call or other unicast audio stream, it may expect to return to the same Broadcast Source, in which case it would not want to remove the Broadcast Receive State characteristic instance. This is an implementation decision, but one which improves the user experience.

Note that there is no state machine for the Broadcast Sink. It uses its Broadcast Receive State characteristics to synchronize with or release broadcast Audio Streams.

8.9 Handovers between Broadcast and Unicast

Two handover procedures are defined in CAP to cover the use cases where an Initiator wants to move between a unicast and a broadcast stream (or vice versa) to transport the same audio content. This is not the same as the general case of changing from a unicast use case to broadcast use case, but is specific to an application such as sharing personal audio with friends, where a user wants to convert from a single, personal stream to a broadcast one, or vice versa.

The procedures in CAP allow an Acceptor to receive concurrent unicast and broadcast Audio Streams from an Initiator, with the intention that the handover could be seamless with no noticeable break in the stream. However, in most cases that will not be supported, as an Acceptor will not have the resources to receive both types of streams at the same time, especially if they are using one of the higher sampling rate QoS configurations. To provide a smooth listening experience, implementations should attempt to conceal any breaks in transmission for the original Acceptor, which may be accomplished by covering them with an audible tone or message, generated locally in the Broadcast Sinks. Friends joining the broadcast Audio Stream will not experience any break, as they would not have been receiving the original audio stream.

The procedures [CAP 7.3.1.10 and 7.3.1.11] also mandate that any Streaming Context Types and CCID_List used for the original stream are carried over to the reconfigured stream. Although the user of the original stream will have transitioned it from unicast to broadcast, the Broadcast Sinks would still maintain their ACL connection with their Audio Source and expect any control functionality to continue, such as fast forward or volume control, hence the need to maintain the CCID association. Others who receive the broadcast will only have access to the audio stream.

Some of the practical issues encountered in unicast / broadcast handovers are covered in Chapter 13.

Chapter 9. Telephony and Media Control

After the complexity of setting up unicast and broadcast streams, the four specifications covering telephony (CCP and TBS) and media control (MCS and MCS) are refreshingly simple, albeit comprehensive. When Bluetooth® technology was first being developed, most of our mobile products were fairly straightforward and just did one thing. Mobile phones made phone calls using the cellular networks and music players played music based on whatever physical media they supported – typically pre-recorded cassettes or CDs. Since then, both telephony and media (by which we mean music and any other audio which we can stop and start) have become a lot more complex.

For telephony, we no longer constrain our phones to a cellular network. The move to Voice over IP (VoIP) has seen an explosion in internet-based telephony services, either as “Over the Top” (OTT) applications on our phones, or as programs on our laptops and PCs. The Covid pandemic and the associated growth in home working accelerated their use. Whilst we still make cellular calls, we also use Skype, Zoom, Teams and a growing host of other telephony services, sometimes using more than one at the same time.

Media has also changed. As we saw in Chapter 1, the growth of Bluetooth Audio has paralleled the growth of music streaming services. We no longer own most of what we listen to, but hire it on demand. That means that the traditional controls of Stop, Play, Fast Forward and Fast Rewind need to be supplemented with new controls which move beyond the physical device to the source of the audio, as features such as variable playback speed need to be applied to the generation of the audio stream itself.

The control mechanisms of Bluetooth Classic Audio profiles have struggled to keep up with this evolution, particularly in telephony, where they are still based on the old “AT” command set, which was first designed for landline modems back in 1981, appearing in the Hayes 300 baud “Smartmodem”⁶⁸. It quickly became a standard for landline modems, before being integrated into the early GSM standards in the 1990s and Bluetooth technology’s Headset and Hands-Free Profiles in the early 2000s. The development of Bluetooth LE Audio has given us the opportunity to go back to first principles for both telephony and media control, with universal state machines which are equally applicable to cellular and VoIP telephony, and all kinds of local and remote media sources.

Content control is currently defined using two sets of profiles and services. In the case of telephony, they are:

- TBS – The Telephone Bearer Service, which defines the state of the device handling

⁶⁸ Although I’ve not been able to verify it, this may also have been the first product to use the prefix “smart”.

the call, and

- CCP – The Call Control Profile, which is the Client acting on TBS.

For Media, the respective pair of specifications are:

- MCS – The Media Control Service, defining the state of the source of the media, and
- MCP – The Media Control Profile, which is the Client acting on MCS.

9.1 Terminology and Generic TBS and MCS features

Up until this point, I've mostly been using the Initiator and Acceptor terminology from CAP as the easiest way of describing what happens to an Audio Stream. Now that we're looking at control, which is orthogonal to the audio streams, that's no longer appropriate. As the basic architecture of Bluetooth LE Audio separates the audio data plane and the control plane, it means that control features can be implemented on devices which don't take part in Audio Streams. Because of that, we need to revert to Client-Server terminology in this chapter, where the Server is the instance of the Service – defining the state of the media player or the phone. For the Telephone Bearer and Media Control Services, the Server is located on the Initiator. The Client can be on the Acceptor, but is equally likely to be in a remote control, simply because that's easier to use, particularly for small devices like earbuds and hearing aids. We've got used to the ease of use of remote controls – it's why they were invented. It doesn't need to look like a conventional remote control - it could be in the form of a smartwatch, another wearable device, or even buttons on an earbud's battery case. Moreover, there can be multiple Clients operating on the Server at the same time. You could accept a phone call on your earbud, but terminate it or put it on hold with your watch.

As we'll see, the control profiles can move us around the media and call states on an Initiator, reflecting the user's desire to start or accept a phone call, or play or pause a piece of music. What they don't do is start or stop any Audio Streams associated with those decisions. The link between control commands and the configuration and establishment of the Audio Streams which transport the audio data is entirely down to the implementation. It is up to applications on the Initiator to tie them together.

In both TBS and MCS, the service specification includes individual TBS and MCS instances, which can be instantiated for each application on the device. For example, if your phone supports Skype, WhatsApp and Zoom, as well as cellular calls, it can include a separate instance of TBS for each one of those applications. If it's a media playback device, it can include an instance of MCS for Spotify, BBC Sounds and Netflix.

However, if you're controlling either telephony or music from a pair of earbuds, with a very limited user interface, you're unlikely to want to, or even be able to discriminate between the different applications. To cope with this, the TBS and MCS specifications each contain a generic version of the service, called the Generic Telephone Bearer Service (GTBS) and

Section 9.1 - Terminology and Generic TBS and MCS features

Generic Media Control Service (GMCS) respectively. These behave in exactly the same way as TBS and MCS, but provide a single interface to the Server device. It is up to the implementation on the server to map incoming commands from a Client to the appropriate application.

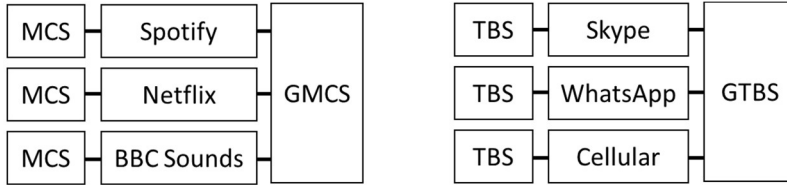


Figure 9.1 Representation of Control Services and their generic counterparts

Figure 9.1 is a simplistic representation of how this works. Every content control Server must include a single instance of the generic service – either GTBS or GMCS, and may have individual instances of TBS and MCS if it wants to expose control directly to an application. You can have as many instances of each service as you have telephony and media applications, but that mapping is down to the implementation.

The respective profiles – CCP and MCP, define Client and Server roles, which are illustrated in Figure 9.2.

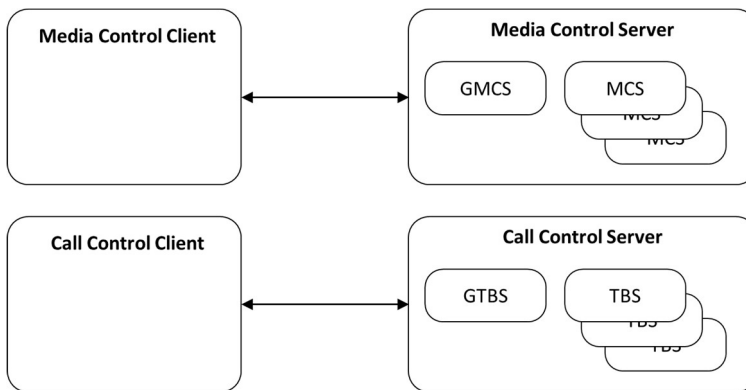


Figure 9.2 Relationship of generic and specific control services

Implementations can use the combination of these services to:

- Treat each application as a unique media player or telephony service,
- Treat the device as a single telephony or media player where commands act on the whole device, or
- A combination, where a subset of commands may be directed to specific applications, being mapped according to their specific Content Control IDs (CCIDs)

In all cases, the mapping is down to the product implementation.

Both TBS and MCS include a wide range of features, which we'll cover below. A lot of the time, Bluetooth LE Audio is concerned with earbuds, as that is currently the largest market by volume. The limited user interface (UI) of an earbud may make readers wonder why so many of these features are included in these specifications and also why so few of them are mandated. That ignores the history of Bluetooth Audio, where, until recently, the major audio application was in carkits, where a very rich control and information interface is provided. Both TBS and MCS were designed to replicate many of the features currently available in carkits, so that Bluetooth LE Audio can be used to build the next generation of these products, as well as extending them to add new features, particularly for the control of media players.

9.2 Control topologies

At the heart of both of the pairs of control specifications is the concept of state, defined in the Service specification. The state for telephony and media control is held on the device where the audio application resides – the media player for MCS, or the telephony device, acting as a gateway to the call bearer for TBS, which is always the Initiator. Client devices can implement the corresponding profile, which writes to a control point on the Server, causing its state to transition. Once that transition has been made, which may also happen locally by the user pressing a button or touching a screen on the Initiator, the Server notifies its new state. It's exactly the same principle we came across in the ASCS state machine, but there is now one device which has the state, while multiple Clients which can read and manipulate it. In the case of both MCP and CCP, the Client can be the device receiving the audio stream, or a device including the Commander role, such as a remote control or a smartwatch. However, in this case the Client in the Commander would be operating on the Initiator, not the Acceptor, as shown in Figure 9.3.

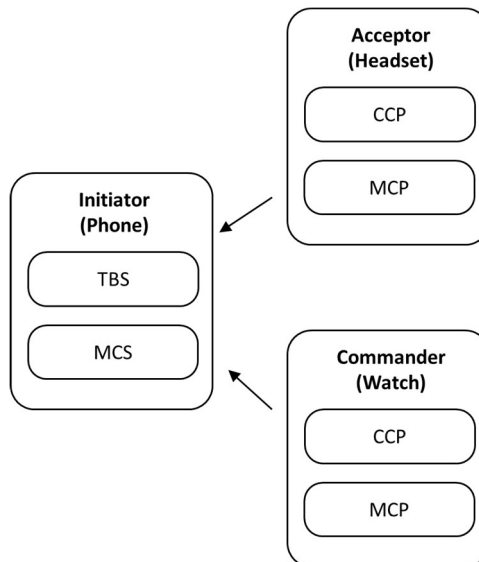


Figure 9.3 Topologies for content control

Section 9.3 - TBS and CCP

In addition to the control point and state characteristics, each service contains a wide selection of characteristics which can be read or notified to determine further information about the application and external service supplying it.

9.3 TBS and CCP

At the heart of the telephony control specifications is a generic call state machine, illustrated in Figure 9.4 and defined in TBS.

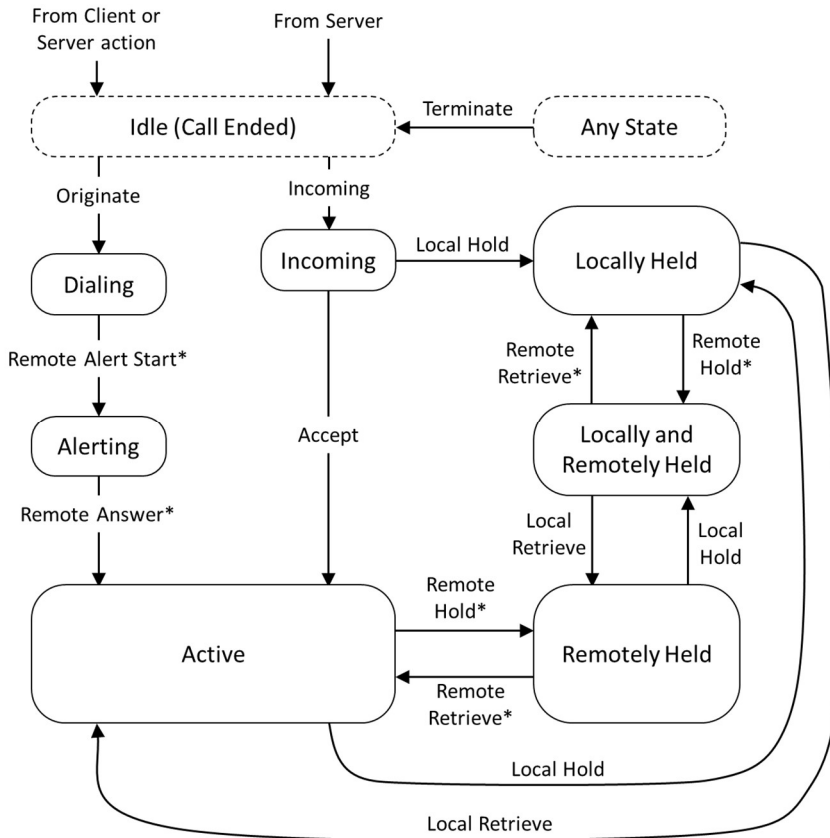


Figure 9.4 The TBS state machine (asterisks denote remote operations)

The heart of the state machine is the Active state, which signifies the presence of a call. In most cases, this state would be attained when a unicast Audio Stream had been established between the Initiator and Acceptor, normally consisting of a bidirectional stream. However, the state machine is equally valid if the call is being taken using a wired headset, or if the phone is being used as a speakerphone with no active Audio Streams. In these cases, the user could use their smartwatch to control the call. This emphasises the distinction between the control plane, which in this case is the TBS / CCP relationship, and the audio data plane, which is the BAP / ASCS relationship. It is up to the application to tie them together as it wishes. (Although they are an integral part of GAF, they are not restricted to Bluetooth LE Audio and could be used with other applications.)

The transitions around the state machine can be prompted by a range of events, such as:

- An incoming call,
- A CCP Client writing to the TBS Call State characteristic,
- A local user action, such as answering a call on the phone, or
- An operation by the remote caller.

The remote operations in Figure 9.4 (Remote Alert Start, Remote Answer, Remote Hold and Remote Retrieve) are marked by an asterisk (*) at the end of each command. These transitions can only be made by the remote caller. The states are exposed by the Call State characteristic, which is notified to Clients whenever the state of the call changes.

9.3.1 The Call State characteristic

The Call State characteristic is an array of all of the current calls on the device. As soon as any call enters a non-Idle state it is assigned a unique, single-octet, sequential index number by TBS, called the Call_Index, with a value between 1 and 255, which is notified in the Call State characteristic. (The Call Control state machine of TBS does not contain an explicit Idle state, but one is shown in Figure 9.4 for clarity). The second octet of the Call State characteristic identifies the current state of each call, and the final octet holds Call Flags associated with that call.

The format of the Call State characteristic is shown in Figure 9.5.

Call_Index [i] (1 octet)	State [i] (1 octet)	Call_Flags [i] (1 octet)
-----------------------------	------------------------	-----------------------------

Figure 9.5 Call State characteristic

Table 9.1 lists the state values which are returned in the Call State Characteristic.

State	Name	Description
0x00	Incoming	An incoming call (normally causing a ringtone).
0x01	Dialing	An outgoing call has started, but the remote party has not yet been notified (the traditional dialtone state).
0x02	Alerting	The remote party is being alerted (they have a ringtone).
0x03	Active	The call is established with an active conversation.
0x04	Locally Held	The call is connected, but on hold locally (see below)
0x05	Remotely Held	The call is connected, but on hold remotely (see below)
0x06	Locally and Remotely Held	The call is connected, but on hold both locally and remotely (see below)
0x07-0xFF	RFU	Reserved for Future Use

Table 9.1 States defined for the Call State characteristic

Section 9.3 - TBS and CCP

9.3.1.1 Local and Remote hold

States 0x04, 0x05 and 0x06 refer to calls which have been placed on hold. These states are differentiated by who it was who put the call on hold. A value of 0x04 indicates that it has been put on hold locally, i.e., by the user with the phone or PC with this TBS instance. A value of 0x05 means that it has been put on hold by the remote party, and a value of 0x06, means that both ends on the call have put it on hold.

A locally held call can be retrieved (brought back to the Active state) by writing to the control point. A remotely held call needs to be retrieved by the remote party. The only local action that can be applied to a remotely held call is to terminate it.

Not all of these states may be supported. For example, many Push to Talk (PTT) telephony applications do not support held calls.

9.3.1.2 Call Flags

The final octet of the Call State characteristic is a bitfield containing information on the call, with the meanings and corresponding values shown in Table 9.2.

Bit	Description	Value
0	Call Direction	0 = incoming call 1 = outgoing call
1	Information withheld by server	0 = not withheld 1 = withheld
2	Information withheld by network	0 = provided by network 1 = withheld by network
3 - 7	RFU	Reserved for Future Use

Table 9.2 Call status bits for the Call State characteristic

Bits 1 and 2 indicate cases where information, such as the URI (typically the caller ID), or Friendly Name may be deliberately withheld because of either a local or network policy. Either or both may be set, in which case the fields of the associated characteristics may be empty or null. Alternatively, the Server can fill them with appropriate text, such as “Withheld” or “Unknown Caller”. The choice of policy and text is implementation specific.

9.3.1.3 UCIs and URIs

Two initialisms you will come across in telephony applications are URI and UCI, which stand for Uniform Resource Identifier and Uniform Caller Identifier. Both are designed to provide call information which covers a wide variety of telephony applications and bearers.

The Uniform Caller Identifier is essentially the bearer; for example, “skype” or “wtsap”. The values for the UCIs are listed in the Bluetooth Uniform Caller Identifiers Assigned Numbers document and are generally abbreviated to no more than five characters. Hence WhatsApp is

“wtsap”. Standard phone numbers use the UCI of “E.164” or “tel:”, signifying a standard dialer format.

The Uniform Resource Identifier is a combination of the UCI and the caller ID, i.e., the caller’s number or username, depending on the application. It can be used for either an incoming call, where it is the Caller ID, or an outgoing call. An application can use the UCI portion of a URI for an outgoing call to select the appropriate telephony application to make the call. URIs are expressed as a UTF-8 string.

9.3.2 The TBS Call Control Point characteristic

As we’ve seen with BAP and ASCS, a Client can move a Server around its state machine by writing to a control point characteristic. In this case, it’s the TBS Call Control Point characteristic. This requires a single opcode, followed by a parameter which depends on the specific opcode, as indicated in Figure 9.6.

Opcode (1 octet)	Parameter (varies)
---------------------	-----------------------

Figure 9.6 The TBS Call Control Point characteristic

Table 9.3 shows the current opcodes and describes their purpose.

Opcode	Name	Parameter	Description
0x00	Accept	Call_Index	Accepts the incoming call.
0x01	Terminate	Call_Index	Terminate the call associated with Call_Index, regardless of its state.
0x02	Local Hold	Call_Index	Places an Active or Incoming call with Call_Index on Local Hold
0x03	Local Retrieve	Call_Index	Moves a locally held call to the Active state.
0x04	Originate	URI	Starts a call to the URI.
0x05	Join	List of Call_index values	Joins the calls identified in the list of Call_Index values.
0x06 – 0xFF	RFU		Reserved for Future Use

Table 9.3 Call Control Point characteristic opcodes for moving around the TBS state machine

The opcodes of Table 9.3 are requests to the Server, which it passes to its relevant telephony application. Some of these: Accept, Terminate and Originate, refer to actions which will be taken locally by the telephony application. Local Hold and Retrieve generally need to be passed back to the network, and Join is almost always a network function. Depending on the specific telephone bearer and application, not all of these functions may be available. They may also fail based on the current phone resources and the network connection. For example,

Section 9.3 - TBS and CCP

some phone services do not allow a call to be dialled if a current call is active. Equally, if there is no cellular signal, an attempt to originate a call will fail.

If the operation is successful, the Call State characteristic is notified, informing the Client of the current State and Call_Index. In the case where the opcode was Originate (0x04), this is the first time the Client will be made aware of the Call_Index

When a Write to the TBS Call Control Point characteristic by a Client fails, the Call Control Point characteristic notifies the result of the opcode write using the following format:

Requested Opcode (1 octet)	Call_Index (1 octet)	Result Code (1 octet)
-------------------------------	-------------------------	--------------------------

Figure 9.7 TBS Call Control Point characteristic notification format

A TBS instance can indicate whether the Local Hold and Join functions are supported by using the Call Control Optional Opcodes characteristic. This is a bit field which a Call Control Client can read to determine whether it should issue these commands. The values in this field are normally static for at least the duration of a session, and generally for the lifetime of the device.

The Result_Code indicates Success, or provides the reason for the failure of the operation. These codes are listed in Table 3.11 of the TBS specification.

9.3.3 Incoming calls, inband and out of band ringtones

In traditional telephony design, the first thing that a user knows about an incoming call is the phone ringing. As soon as you introduce a Bluetooth connection, this becomes more complicated. The phone can still audibly ring, or the ring may happen in your earbuds (if you're wearing them). Or it can happen on both.

There is a further complication. The ringtone that you hear in your earbuds may be identical to the sound on your phone, or it may be a locally generated ringing tone produced by the earbud, which will be different. If it's the same as the sound of your phone, it requires the presence of an Audio Stream from the phone, which carries the same ringtone audio that is being played on your phone's speaker. That's called an inband ringtone. The problem with an inband ringtone is that you need to set up the Audio Stream to carry it, which may mean tearing down an existing Audio Stream to another device. Imagine the use case where you're using your earbuds to listen to your TV and your phone rings. For an inband ringtone, your earbuds will probably need to terminate the Audio Stream with the TV and replace it with an Audio Stream from your phone. If you accept the call, that's not a problem, but if you reject it, you will need to reconnect to the TV. The resulting breaks in the audio stream are a poor user experience, which the use of an inband ringtone may not be ideal.

The alternative is for the phone to notify its Incoming Call characteristic to your earbuds. This

contains the `Call_Index` and `URI`, i.e., the `URI` scheme and the `Caller ID` of the incoming call. The `Call Control Client` in your earbud can generate a local ringing tone alerting you to the call. If you accept the call, the `Call Control Client` will write an `Accept` opcode (0x00) to the phone's `Call Control Point` characteristic. That instructs your earbud to terminate the `Audio Stream` with the `TV`, at which point the phone (which is a different `Initiator`) can establish a bidirectional `Audio Stream` to carry the call. At the end of the call, the phone application should ask if you wish to reconnect to the `TV`.

If you reject the call, your `Audio Stream` with the `TV` remains, as it has not been interrupted – your earbud will simply have mixed its locally generated ringtone with the `TV` audio. It's a much cleaner user experience, but it does mean that your earbuds will make a different ringing sound to your phone.

If your phone is also the source of your media content, the phone can mix the appropriate audio streams. The issue arises when two different `Initiators` are involved.

The use of out of band ringtones can be enhanced if the earbud can perform text-to-voice conversion, allowing it to speak the phone number contained in the `Incoming Call` characteristic as part of the call alert. If the phone supports the optional `Friendly Name` characteristic, which contains the text of the caller name from your contact list on the phone, this could also be spoken within the earbud.

One final subtlety to call alerts is that the phone can be set into silent mode, where an incoming ringtone is not sent to the phone (or PC's) speaker, but where the announcement is left to the `Acceptor`. A `Call Control Client` can control this, along with whether the phone supports an inband ringtone, by reading the `Status Flags` characteristic, shown in Table 9.4.

Bit	Description	Value
0	Inband Ringtone	0 = disabled 1 = enabled
1	Silent Mode	0 = disabled 1 = enabled
2 - 15	RFU	Reserved for Future Use

Table 9.4 Status flags characteristic for ringtone mode support

If the `Inband ringtone` is disabled, the `Call Control Client` would normally announce an incoming call at the point when it is notified that the `Call State` characteristic has transitioned to the `Incoming` state. Note that this includes `Call Control Clients` which are not `Acceptors`. For instance, a smart watch that included a `Call Control Client` might vibrate or ring, although it could not accept an `Audio Stream`. This highlights the fact that the `Call Control` state machine has no immediate connection to an `ASE` state machine. One does not directly trigger the other. It is entirely up to the telephony application or operating system to link the call control states to `Audio Stream` management.

Section 9.3 - TBS and CCP

If the Status Flags characteristic shows that Silent Mode is enabled, it means that the ringtone will not be played on the phone. Depending on the state of the Inband Ringtone (bit 0), it may be sent to the Acceptor using an Audio Stream, or as an out of band ringtone, or both. If both are enabled, it is up to the Acceptor to decide which to use.

It is worth remembering that accepting or rejecting the call, along with all other state transitions, can be performed on the Call Control Server (i.e., the phone or PC) as well as on a Call Control Client. All connected Call Control Clients have equal access; any action that changes the state will generate a notification.

Designers need to think carefully about the user experience of presenting ringtones. It is important to a user that the notification of an incoming call is consistent, both in terms of where the audible ringtone is generated (phone or headset), the delay between it appearing on a phone and the headset, and the sound of the ringtone, which may be different for inband and out of band settings. These need to be considered for the case where the same phone is responsible for media playback and telephony, as well as when different devices are involved, both of which may support both media playback and telephony.

9.3.4 Terminating calls and failed calls

A call can be terminated by any Call Control Client, by a user action on the phone, or by the remote caller. It can also be lost for a variety of reasons, such as a fault or loss of signal on the telephone bearer network. Whenever a call is terminated, the Call Control Server will use the Termination Reason characteristic to notify the Call Control Clients of the reason for the termination. Like similar characteristics, it includes the Call_Index to identify the call and the termination reason, as shown in Figure 9.8.

Call_Index (1 octet)	Reason_Code (1 octet)
-------------------------	--------------------------

Figure 9.8 Termination Reason characteristic

If multiple calls are terminated, as might happen in the case of a network issue on a joined call, then a separate Termination Reason notification is sent for each Call_Index. The termination reasons are shown in Figure 9.5. There are also a set of reason codes covering dialled calls which fail to connect. Applications may use these to prompt a subsequent user action.

Reason_Code	Reason
0x00	Incorrectly formed URI for originating call
0x01	The call failed
0x02	The remote party ended the call
0x03	The server ended the call
0x04	Line busy
0x05	Network congestion
0x06	A client ended the call
0x07	No service
0x08	No answer
0x09	Unspecified
0x0A – 0xFF	Reserved for Future Use

Table 9.5 Termination Code characteristic Reason_Code values

9.3.5 Other TBS characteristics

As mentioned above, TBS contains a large number of other characteristics for the Call Control Server role. These characteristics provide more detailed information about the phone call. They can be read by more complex clients, such as carkits, which can use them to replicate the user experience of the phone by mirroring the level of information which is available from the original telephony application.

Table 9.6 provides a list of the additional characteristics which have not been covered above. All of them are mandatory for a GTBS or TBS instance to support, with the exception of the Bearer Signal Strength characteristic and its dependent Bearer Signal Strength Reporting Interval characteristic. They are all described in the Telephone Bearer Service specification.

Characteristic Name	Description
Bearer Provider Name	Telephony service. E.g., Vodafone, T-Mobile
Bearer Uniform Caller Identifier	UCI, e.g., skype, wtsap, from Assigned Numbers
Bearer Technology	As displayed on the phone. E.g., 2G, 3G, Wi-Fi
Bearer Signal Strength (<i>Optional</i>)	Signal strength from 0 (no signal) to 100
Bearer Signal Strength Reporting Interval	How often the signal strength is reported (<i>Mandatory if Bearer Signal Strength is supported</i>)
Bearer URI Schemes Supported List	A list of supported URI schemes
Bearer List Current Calls	A list of all current calls and their state
Content Control ID (CCID)	A CCID value which remains static until a service change
Incoming Call Target Bearer URI	The incoming URI of the call. E.g., your phone's number.

Table 9.6 Other characteristics specified in TBS

There are corresponding procedures to read all of the TBS characteristics in the Call Control Profile. Although support for most of the characteristics in TBS are mandatory, the only Call Control procedure which is mandatory is the Read Call state procedure. This reflects the fact that for devices with a constrained user interface, such as hearing aids and earbuds, most control actions are likely to take place on the phone.

9.4 MCS and MCP

Once you’ve understood telephony control, the media control specifications will look very familiar. The Media Control profile defines two roles – the Media Control Client and the Media Control Server. The latter resides on an Initiator, where the Media Control Service defines a state machine for media playback, (which is shown in Figure 9.9), along with a plethora of characteristics to notify what it’s doing.

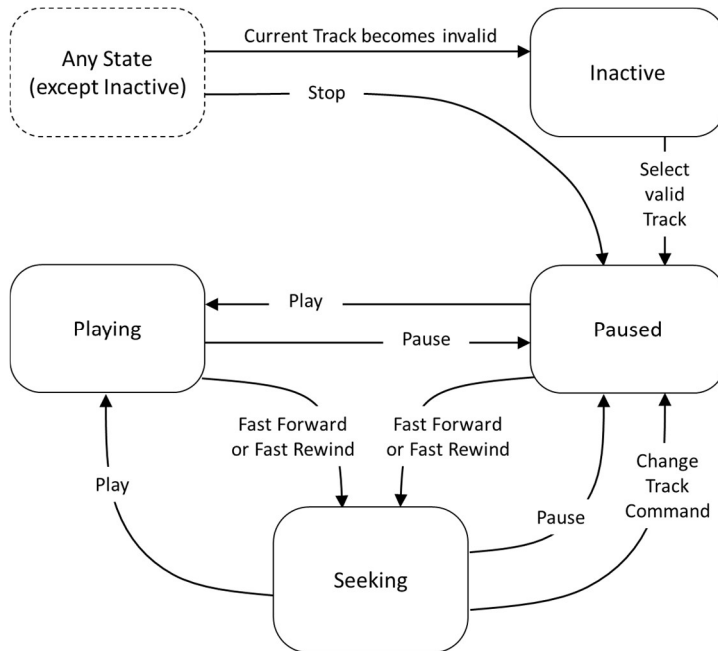


Figure 9.9 MCS state machine for media control

For Media, the player normally resides in the Inactive state, moving to the Paused State when a Track is selected, from which it can be transitioned to Playing. From Playing, it can return to the Paused state by stopping it, or be moved to the Seeking state by issuing a Fast Forward or Fast Rewind command. There is no Stop state in the state machine – a Stop command in any state will move it to the Paused state. There is a subtle distinction, in that the Stop opcode will return to track position to the start of the current track, whereas the Pause opcode will retain the track position when that command was executed.

Other than the different effect on track position, there is little real distinction between the Stop and Pause commands. That difference harks back to analogue devices where Stop turned off the motor of a record or cassette deck, whereas Pause left the motor running and lifted the stylus or playback head off the playback medium. When everything is in digital memory, the operations are more similar.

9.4.1 Groups and Tracks

The three main states – Paused, Playing and Seeking are only valid when there is a current track selected, which brings us to the concept of tracks and groups, which is a key part of MCS. Figure 9.10 illustrates the concept, showing a hierarchy of Groups, which contain Tracks. It is entirely up to an implementation to define how these are structured, but a Group is probably best visualised as a traditional album, where the tracks are individual songs. Groups can be nested to become parts of larger groups, so a Parent Group consisting of the entire works of Beethoven might contain subgroups of Orchestral music, Chamber music, Piano, Vocal works, etc., which could each contain further collections of subgroups. All that MCS requires is that there is a defined structure to create a playing order from the start of the first group to the end of the last group.

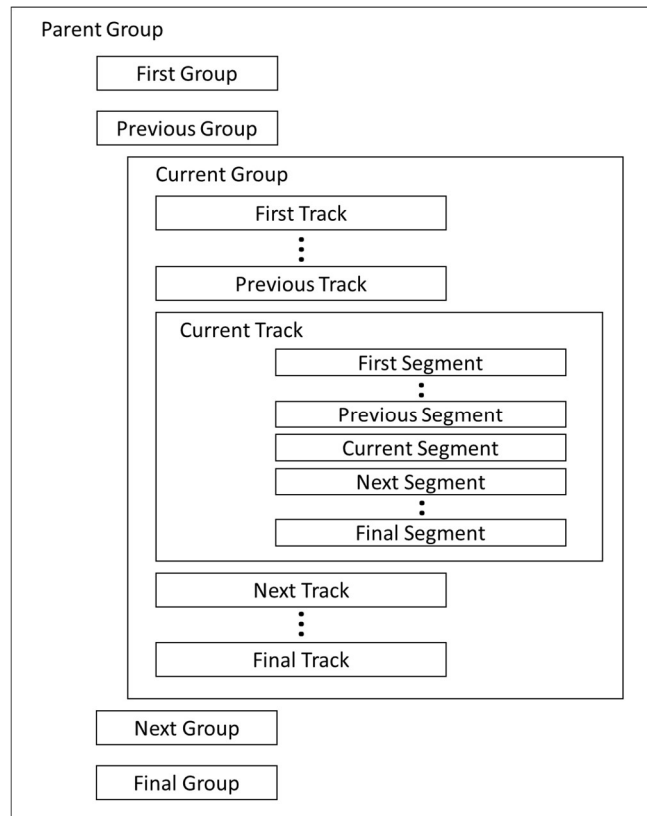


Figure 9.10 MCS' arrangement of Groups and Tracks

Section 9.4 - MCS and MCP

Groups contain tracks, which may also contain segments. Most of the MCP controls operate on tracks, which are what most people would regard as conventional tracks on a record or CD. Specifically, the operations of the state machine focus on the Current Track. The key elements of a Track are shown in Figure 9.11, which are the Track Duration, Offsets from either end of the Track and the Playing Position.

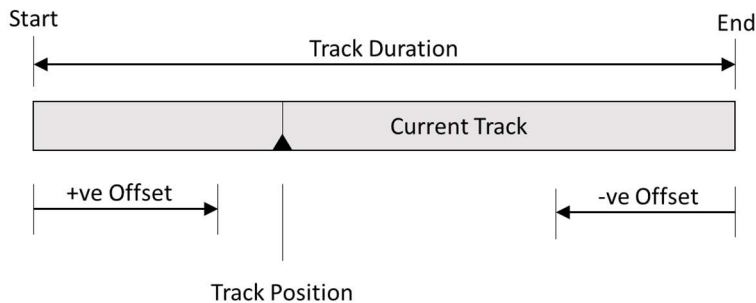


Figure 9.11 Key elements of a Track

9.4.2 Object Types and Search

MCP and MCS include a lot of functionality to support complex user interfaces, such as the screen of a car's AV system. Most of this is based on the existing Bluetooth LE Object Transfer Service (OTS) which allows extended information to be associated with Groups and Tracks. This includes extended names, icons and URLs which can be used to fetch more complex objects, such as album covers. OTS is also used for returning search results. MCS allows some very comprehensive search functions. You can search using combinations of Track Name, Artist Name, Album Name, Group Name, Earliest Year, Latest Year and Genre. For more information on these, look at the MCS specification. They're unlikely to be used in products in the short term, so I'll skip the detail and concentrate on the key aspects of media control.

9.4.3 Playing Tracks

With the structure of Groups and Tracks explained, we can look at how media control works. Once a track is selected, which is generally a user action on the media player, the player will notify all of its connected Clients using the Track Changed characteristic. This characteristic has no value, i.e., it is simply a statement that the media player now has a current track, so a Client can start to control it. A Client can read the Track Title Characteristic and the Track Duration characteristic. It can also read the Track Position, which is returned with a 10msec resolution from the start of the track. The maximum track duration allowed is just over 16 months⁶⁹.

⁶⁹ This might seem ridiculous, but the longest single video, "Level of Concern" by the group "twenty

A Client can now write to the Media Control Point characteristic with the values shown in Table 9.7.

Opcode	Name
0x01	Play
0x02	Pause
0x03	Fast Rewind
0x04	Fast Forward
0x05	Stop

Table 9.7 Media Control Point characteristic opcodes for media control

At the point that the Current Track starts playing, the Server should notify the Track Position characteristic, to inform Clients of the current track position. The frequency of subsequent notifications is down to the implementation.

Another set of opcodes are available for moving around the current track, which are shown in Table 9.8.

Opcode	Name	Parameters
0x10	Move Relative	32 bit signed integer
0x20	Previous Segment	None
0x21	Next Segment	None
0x22	First Segment	None
0x23	Last Segment	None
0x24	Goto Segment	32 bit signed integer

Table 9.8 Media Control Point characteristic opcodes for in-track movement

The Media Control Service allows segments to be defined within a track. Each segment includes a name and an absolute offset from the start of the track. The commands in Table 9.8 allow a move to a specific segment of the current track. None of these allow movement outside the current track. If the parameter value results in moving outside the current track, then the result will be limited to moving the position to the start of the track or the end of the track, but the track remains the current track.

A track becomes invalid when the end of the current track is reached, with no following track in the playlist. Issuing the Stop command does not result in the current track becoming invalid,

one pilots” is 177 days, 16 hours, 10 minutes and 25 seconds long, and it is likely that someone will try to break this record.

Section 9.4 - MCS and MCP

as the track position is set to the start of the current track.

Two further sets of opcodes allow a different track to be selected as the current track.

Move within a Group			Move to another Group		
Opcode	Name	Parameters	Opcode	Name	Parameters
0x30	Previous Track	None	0x40	Previous Group	None
0x31	Next Track	None	0x41	Next Group	None
0x32	First Track	None	0x42	First Group	None
0x33	Last Track	None	0x43	Last Group	None
0x34	Goto Track	32 bit signed INT	0x44	Goto Group	32 bit signed INT

Table 9.9 Media Control Point opcodes for moving around Tracks and Groups

When another Group is selected, the first track of the current group resulting from the command becomes the current track.

All segments, tracks and groups start numbering at zero. Where a Goto operation is used, a positive value “n” is equivalent to going to the first item and then executing the Next Track opcode n-1 times. A negative value is equivalent to going to the last item and executing the Previous Track opcode n-1 times. Moving around segments is analogous.

Moving to a new track or segment makes it current. Whether the resulting state is Playing or Paused is left to the implementation.

It is only mandatory to support a minimum of one of the opcodes in the Media Control Point characteristic. It is unlikely that any implementation would be that spartan, but Clients can check what the Server supports by reading the Media Control Point Opcodes Supported characteristic. This is a bitfield which indicates which opcodes the Server supports.

As always with Control Point opcodes, a Client writes the opcode and receives a notification of the operation in the Media Control Point characteristic, with a Result_Code indicating Success, Opcode not supported, or Media Player Inactive. Once the Server has fulfilled the request it will also notify the appropriate characteristic for that operation if a change has occurred in its value.

9.4.4 Modifying playback

The Media Control Service has a couple of neat features for modifying playback and seeking. The first of these is the ability to request a change in playback speed, using the Playback Speed characteristic. This can be written by the Client to request that the current track is played faster or slower. The parameter for the characteristic is an 8-bit signed integer “p”, which

ranges in value from -128 to 127, where the actual playback speed requested is:

$$speed = 2^{\frac{p}{64}}$$

I.e., the speed is 2 to the power of (p/64). This gives a range of 0.25 to 3.957, which is effectively ¼ to 4 times the standard speed.

Playback speed is a blind request; the Client does not know whether or not the value of “p” is supported. After the request, the Server notifies the value which has been set in the Playback Speed characteristic. If the request was outside the range that the Server supports, it should set it to the closest available speed. If the Media source is live or streamed, then the Playback Speed characteristic value may be fixed to a value of 1. It is entirely up to the implementation to decide how to adjust the speed and whether to maintain the pitch of the Audio Data.

The other characteristic which can be varied is the Seeking Speed, by writing a signed integer value to the Seeking Speed characteristic. This is used as a multiple of the current Playback speed, with positive values signifying Fast Forward and negative values Fast Rewind. The Seeking Speed value is applied to the current Playback Speed, not the default Playback Speed, where the value of p would be 0. A value of 0 for the Seeking Speed indicates a Seeking Speed which is the same as a Playback Speed with a value of p = 0, regardless of the current Playback Speed.

During Seeking, the Track Position characteristic should be notified to provide an indication of the current Track Position. The frequency of notification is determined by the implementation. Seeking will stop when the Track Position has reached the start or end of the current track.

Whilst seeking, MCS states that no Audio Data should be played. However, the Context Type, which describes the use case (not the content of the Audio Stream) would not normally change.

9.4.5 Playing order

The last feature to cover in MCS is the playing order. The Playing Order characteristic provides ten different ways to play tracks. Most of them cover the order of play when you’re playing an entire Group. The options are listed in Table 9.10.

Value	Name	Description
0x01	Single once	Play the current track once
0x02	Single repeat	Play the current track repeatedly
0x03	In order once	Play all of the tracks in a group in track order
0x04	In order repeat	Play all of the tracks in a group in track order, then repeat
0x05	Oldest once	Play all of the tracks in a group once, in ascending order of age
0x06	Oldest repeat	Play all of the tracks in a group in ascending order of age, then repeat
0x07	Newest once	Play all of the tracks in a group once, in descending order of age
0x08	Newest repeat	Play all of the tracks in a group in descending order of age, then repeat
0x09	Shuffle once	Play all of the tracks in a group once, in random order
0x0A	Shuffle repeat	Play all of the tracks in a group once, in random order, then repeat.

Table 9.10 *Playing Order characteristic values*

There are a couple of nuances in these. The first two values, which play the current track (0x01 and 0x02) set the current track to the next track when they have completed or been Stopped. For shuffle, the randomisation is left to the implementation. In most cases the implementation is not totally random, but weighted, as listeners do not expect the same tracks to be played close together at the start of a subsequent cycle. Nor is it stated whether the shuffle repeat is in the same order as the first random order. These decisions are left to the implementation. The Client simply sends the command. If the Server cannot support the requested playing order, for example when it doesn't know the age of the Tracks, it should ignore the command.

As with the Media Control Point characteristic, a Client can determine which Playing Order features are supported by reading the Playing Orders Supported characteristic. This is a 2-octet bitfield with the bitfield meanings shown in Table 9.11

Value	Name
0x0001	Single once
0x0002	Single repeat
0x0004	In order once
0x0008	In order repeat
0x0010	Oldest once
0x0020	Oldest repeat
0x0040	Newest once
0x0080	Newest repeat

Value	Name
0x0100	Shuffle once
0x0200	Shuffle repeat

Table 9.11 Meaning of bits in the Playing Orders Supported characteristic

That's it for telephony and media control. Our next stop is volume, audio input and microphone.

Chapter 10. Volume, Audio Input and Microphone Control

Anyone who has worked on audio specifications will probably tell you that a large part of their time was taken up with discussions about volume control. It's a topic which generates even more debate than audio quality. The reason for that is twofold. The first is a never-ending, and generally academic discourse on the perception of volume and how to define the steps between minimum and maximum. The second is how to cope with multiple different ways of controlling the volume. The second problem didn't exist when you had a single volume knob on your TV or amplifier. However, as soon as you have multiple remote controls, it became increasingly challenging for a user to work out how to set the volume.

After the requisite hundreds of hours of debate, the Bluetooth® LE Audio working groups decided to more or less skip the first problem and concentrate on the second – how to coordinate multiple different points of control. So, this chapter is all about volume control, with only a passing reference to how volume is interpreted, because that's left to the implementation. We'll also cover microphone control, as it's analogous, but involves the input of audio, rather than its output.

All of these features are designed to be used on non-encoded audio signals. In the case of volume, that means after reception and decoding; for the Microphone control service and profile, it is before transmission. All of the state resides on the Acceptor.

10.1 Volume and input control

After the previous chapters, this should be plain sailing. There are three services involved with volume and one profile:

- The Volume Control Service (VCS) which can be viewed as the main volume control knob
- The Volume Offset Control Service (VOCS), which can be considered as a “Balance” control
- The Audio Input Control Service (AICS), which allows individual gain control and mute on any number of audio inputs, which don't need to be Bluetooth Audio Streams, and
- The Volume Control Profile (VCP), which lets multiple Clients control these Services.

Although most of these have “volume” in their name, what they actually do is adjust the gain, i.e., the amplification of the audio signal. Note that there is no mechanism to adjust the Source volume. Everything is applied to the audio streams at the Acceptor. As explained in Chapter 2, it is assumed that the input to the codec will be set at “Line” level to ensure the best dynamic range entering the LC3 encoder.

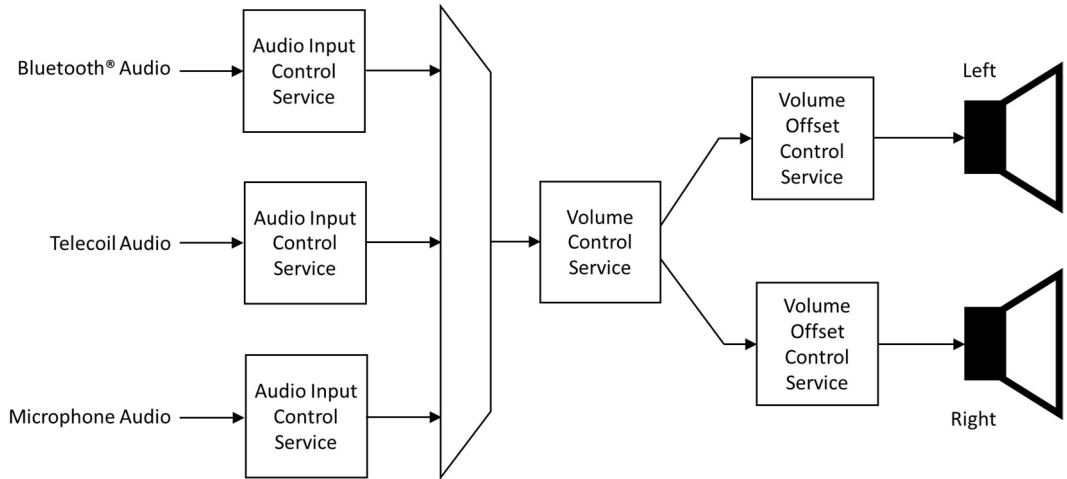


Figure 10.1 Typical implementation of Volume and Audio Input services for a headphone

Figure 10.1 shows a typical implementation for a pair of headphones, or banded hearing aids, illustrating how those services can be combined. The hearing aid has three audio inputs – a Bluetooth Audio Stream, a telecoil audio input and a microphone input. Each of these inputs use a separate instance of the Audio Input Control Service to set their individual gains and to selectively mute and unmute them. The resulting streams are fed to the main gain stage, which has its gain controlled by the Volume Control Service setting. If the stream is a stereo one, then, once it is split into its left and right components, a separate instance of the Volume Offset Control Service can adjust the gain going to each speaker. (The drawing is a hardware representation. In practice the sum of VCS and VOCS gain settings are applied to each Audio Channel.) Used together differentially, these mimic the effect of a balance control. They can also be used individually to adjust the relative level of sound in individual earbuds and hearing aids to accommodate different levels of hearing loss in the left and right ear. There’s an example of that in Figure 10.3.

10.1.1 Coping with multiple volume controls

From the outset, during Bluetooth LE Audio development, it was obvious that users would want to control the volume of their earbuds and headset from multiple different places. The assumption was that there would be local controls on the earbuds, volume controls on the audio sources (typically the phone), as well as separate volume controls in smart watches and other dedicated remote control devices.

To make this level of distributed control work, you want to keep the primary volume control at the device which is rendering the audio. If you don’t, but let it be controlled at the audio source as well, you can end up with one Controller operating on the source level and one on the sink level. That leads to the situation where the source gain gets turned down, with the sink gain turned up to full to compensate. It means that the user can no longer increase the volume at their ears, and the audio quality is reduced, as the codec is sampling and encoding

Section 10.2 - Volume Control Service

a signal with a very low amplitude. This decreases the all-important signal to noise ratio, as on decoding and amplification the noise gets amplified as well. It also helps to get over the problem of amplifying an audio signal at its source, as moving to a different audio source can result in an unexpected sound level at the ear if the sources have different local volume settings.

To address this, it's important that the gain occurs at the end of the audio chain, but for that to work, all of the volume control Clients need to keep track of its value, so they always know where they are. It's a bit like the old joke, where a driver stops to ask a passer-by if he could give him directions to get to Dublin, and the passer-by replies, "If I was going to Dublin, I wouldn't start from here". Fortunately, the GATT Client-Server model has notifications, so we can make use of these to ensure that every Volume Control Client knows the current gain level on the Server. But to be absolutely sure, the Volume Control Service includes an additional safeguard called the Change_Counter, which we'll explore below.

Looking back at Figure 10.1, it also explains the architecture, where the Volume Control state is as close as possible to the final point of rendering. This brings us on to the individual Volume Control Services. These are very thin documents, so we can skim through them fairly quickly.

10.2 Volume Control Service

Like the control service we've seen before, all of the hard work is done by two characteristics:

- The Volume Control Point, which the Clients use to set the volume level, and
- The Volume State, which the Server uses to notify the current volume level after the Volume Control Point characteristic has been written.

We should strictly be talking about gain, rather than volume, but as everyone is used to the colloquial use of volume, I'll stick with it. A third characteristic – Volume Flags, is used to let a Client know about the history of the current volume setting.

The Volume State characteristic has three fields, all one octet long, which are shown in Table 10.1. There is only one instance of the Volume State characteristic on an Acceptor.

Field Name	Values
Volume_Setting	0 to 255. 0 = minimum, 255 = maximum
Mute	0 = not muted 1 = muted
Change_Counter	0 to 255

Table 10.1 The Volume State characteristic of VCS

The `Volume_Setting` is defined as a value from 0 (minimum volume) to 255 (maximum), with no stipulation on how those figures are related to the output volume. It is left to the implementer to map these values to actual output volume, with the expectation that the user perception is approximately linear. I.e., if the `Volume_Setting` value is set to 127, then the resulting output should sound as if it is halfway to the maximum.

This brings us back to the debate about how volume is perceived. Our hearing is generally logarithmic, which is why sound intensity is measured in decibels, but that can be affected by what it is we're listening to. Leaving it to the manufacturer does mean that if speakers or earbuds from different manufacturers are used together, changes to their volume may not align with the listener's expectation. One may appear louder than the other at certain settings. Solving that fell into the "too difficult" category, so it's been left to implementation. Using VOCS as a balance may help that, but if the full scale mapping differs in the devices, there may still be discrepancies.

The value of the `Volume_Setting` field is changed as a result of an operation on the Volume Control Point characteristic, which modifies the current value, or a local operation on the user interface of the device. The value stored in `Volume_Setting` is not affected by any Mute operation.

The Mute indicates whether the audio stream has been muted. This is independent of the `Volume_Setting` value, so that when a device is unmuted, the audio volume will resume at its previous level. An implementation should ensure that any change of volume written to any VCS or VOCS characteristic should result in all devices in that Coordinated Set being unmuted.

`Change_Counter` is the feature mentioned above, which ensures that all Volume Control Clients are synchronized. When an Acceptor implementing VCS is powered on, the Server initialises `Change_Counter` with a random value between 0 and 255. Every subsequent operation on the Volume Control Point characteristic or the device's local controls, whether to change volume or mute, which results in a change to the `Volume_Setting` or Mute field values, will result in the `Change_Counter` incrementing its value by 1. Once it reaches a value of 255, it will roll around to 0 and keep incrementing.

The purpose of `Change_Counter` is to ensure that any Client attempting to change the current volume or mute settings is aware of their current value, to prevent any unexpected changes or conflicts. This is checked by requiring every write procedure by a Client to include the current value of `Change_Counter` which they hold. Unless that matches the value in the Volume State Characteristic, it is assumed they are out of sync, in which case the command is ignored, and no notification will be sent. The Client should recognise the lack of a notification as an error, read the Volume State characteristic and try again. A volume controller that has a visual display of the current volume may want to read these values regularly to maintain a display of the current gain level(s).

Section 10.2 - Volume Control Service

To change the volume, or mute a device, the Volume Control Client writes to the Volume Control Point characteristic of the Volume Control Service using one of the Volume Control Point procedures. This command contains an opcode, a Change_Counter value and, in the case of the Set Absolute Volume command, a Volume_Setting value. The six opcodes are listed in Table 10.2.

Opcode	Operation	Operands
0x00	Relative Volume Down	Change_Counter
0x01	Relative Volume Up	Change_Counter
0x02	Unmute / Relative Volume Down	Change_Counter
0x03	Unmute / Relative Volume Up	Change_Counter
0x04	Set Absolute Volume	Change_Counter, Volume_Setting
0x05	Unmute	Change_Counter
0x06	Mute	Change_Counter
0x07-0xFF	Reserved for Future Use	

Table 10.2 Volume Control Point characteristic opcodes

Most of these are self-explanatory. Relative Volume Down and Up change the volume setting, without affecting the Mute state, so can be used to change the volume level which will be applied when a device is eventually unmuted. Opcodes 0x05 and 0x06 unmute and mute without affecting the volume level, whilst opcodes 0x02 and 0x03 apply a relative volume step at the same time as muting or unmuting. Set Absolute Volume takes an additional parameter, which is the absolute volume level required.

Although the Volume_Setting ranges from 0 to 255 for all devices, very few are likely to contain more than twenty-one⁷⁰ discrete volume levels. This requires a Server to define a Step Size, which is applied whenever a volume setting command is issued. The Step Size is effectively $256 \div \text{Number of steps}$. The following equations are used by the Server to calculate the new value written into the Volume_Setting value of the Volume State characteristic.

Relative Volume Down: $\text{Volume_Setting} = \text{Max}(\text{Volume_Setting} - \text{Step Size}, 0)$

Relative Volume Up: $\text{Volume_Setting} = \text{Min}(\text{Volume_Setting} + \text{Step Size}, 255)$

10.2.1 Persisting volume

The final characteristic is the Volume Flags characteristic, which is a bitfield, of which only one bit is defined. This is Bit 0, which is the Volume_Setting_Persisted Field. If set to 0, it directs a Client to reset the volume by writing to the Volume Control Point characteristic with

⁷⁰ Assuming a 0 to 10 scale, with a resolution of 0.5. Unless you're a Spinal Tap fan, in which case it's twenty-three.

an opcode of 0x04. This sets an absolute volume, which may be a default value determined by an application or the Client device. It may also be the value remembered from the last time that application was used.

If the value is 1, it indicates that the Volume Control Server still has the Volume_Setting from its last session, which should continue to be used for this session. The Volume Client should retrieve this value, either through a notification or by reading it, and use it as the initial value for this session. Although this can reflect any change in the input audio level in the Initiator, it can improve the user experience, by starting with the same volume settings which have previously been used.

10.3 Volume Offset Control Service

If all you have is a single, mono speaker, then the Volume Control Service is all you need. As soon as you are dealing with more than one audio stream, whether that's going to a single Acceptor or multiple Acceptors, you probably want to include an instance of the Volume Offset Control Service for every rendered Audio Location. All of its characteristics are accessed using procedures in the Volume Control Profile.

The Volume Offset Control Service includes four characteristics, shown in Table 10.3. All four of them are mandatory.

Characteristic	Mandatory Properties	Optional Properties
Volume Offset State	Read, Notify	None
Volume Offset Control Point	Write	None
Audio Location	Read, Notify	Write without response
Audio Output Description	Read, Notify	Write without response

Table 10.3 Volume Offset Control characteristics

The Volume Offset State and Volume Offset Control work in the normal manner. The Volume Offset State includes two fields – Volume_Offset and Change_Counter

Field	Size
Volume_Offset	2 octets
Change_Counter	1 octet

Table 10.4 The Volume Offset characteristic

The Volume_Offset is two octets long, as it allows values from -255 to 255. The value of the Volume_Offset is added to the value of Volume_State to provide a total volume value for rendering. The specification does not define what should happen when a combination of VCS and VOCS values exceed the maximum or minimum rendering value for one of the Acceptors, when the other still has room for adjustment. It is left to the implementation to decide

Section 10.4 - Audio Input Control Service

whether further changes are accepted by the Acceptors.

The `Change_Counter` is the same concept we saw in the Volume Control Service and is used to ensure that any Client issuing a Volume Offset Command is aware of the current status of the Volume Offset. Note that although it is the same concept and has the same name, it is a separate instance. Every instance of VOCS, as well as the single instance of VCS, maintains its own value of `Change_Counter`, which is updated when there is any change to their `Volume_Offset` or `Volume_State` field value.

To effect a change, a Client writes to the Volume Offset Control Point characteristic with the following parameters:

Parameter	Size (octets)	Value
Opcode	1	0x01 = Set Volume Offset
Change_Counter	1	0 to 255
Volume_Offset	2	-255 to 255

Table 10.5 The Set Volume Offset parameters (opcode = 0x01)

10.3.1 Audio Location characteristics

The Volume Offset Control service contains two characteristics relating to the Audio Location to which the offset is applied.

The first of these is the Audio Location characteristic. This is a 4 octet bitmap which defines which Audio Location the offset should be applied to, using the format of Audio Locations from the Generic Audio assigned numbers. Normally, this will contain a single location, as a separate VOCS instance is applied to each Audio Location. This is normally a read-only characteristic, set at manufacture, but it can also be made writable. Some devices are able to change their Audio Location, either locally or by a Client action. The specification does not define how this is handled in VOCS.

The Audio Output Description characteristic allows a text string to be assigned to each Audio Location to provide more information for an application, e.g., Bedroom Left Speaker. It may be assigned at manufacture, or made accessible for users to assign friendly names.

10.4 Audio Input Control Service

The final part of the rendering trio of services is the Audio Input Control Service (AICS). This acknowledges that many products, such as hearing aids, earbuds and headphones, contain more than one source of audio streams and that more than one of them may be used at the same time. The most common example is the concurrent use of an external microphone receiving ambient sound, along with a Bluetooth stream. For hearing aids, this has always been the way they work. More recently, with the appearance of transparency modes, it has become an increasingly popular feature in earbuds and headphones, so that wearers can be

made aware of sounds from the world around them whilst listening to a Bluetooth stream.

An instance of the Audio Input Control Service is normally included for each separate audio path which is intended to be mixed and rendered on a device. If there is a single Bluetooth LE Audio path, it offers no advantages over VCS, so does not need to be included. It can also be used with the Microphone Control Service to provide features where audio is being captured. This is covered in Section 10.7.

Audio inputs which are not directly fed to the output, such as microphones which provide inputs for noise cancellation, would not have their own instance, as they would be used for processing another audio stream (which might have its own AICS instance).

The Audio Input Control Service contains six characteristics, which are listed in Table 10.6. All of them are mandatory to support.

Name	Mandatory Properties	Optional Properties
Audio Input State	Read, Notify	None
Audio Input Control Point	Write	None
Audio Input Type	Read	None
Audio Input Status	Read, Notify	None
Audio Input Description	Read, Notify	Write without response
Gain Setting Properties	Read	None

Table 10.6 Audio Input Control Service characteristics

Before jumping into these, we need to understand how AICS defines Gain. With volume, VCS and VOCS simply define a linear scale from 0 to 255, much like a knob on a Hi-Fi unit which goes from 0 to 10, and leaves it up to the manufacturer to convert that to what is usually a decibel based, internal mapping. With AICS, the assumption is made that gain will be decibel based, so the figures for Gain are in decibel-based units. However, to provide a bit more flexibility, the actual step in gain values can be defined in multiples of 0.1 dB. What those multiples are is a manufacturer specific choice, which is exposed in the Gain Setting Properties characteristic, along with minimum and maximum permitted values, as shown in Table 10.7.

Name	Size (octets)	Description
Gain_Setting_Units	1	The increment, in 0.1db steps, applied to all Gain_Setting values
Gain_Setting_Minimum	1	The minimum permitted value of Gain_Setting
Gain_Setting_Maximum	1	The maximum permitted value of Gain_Setting

Table 10.7 Fields of the Gain Setting Properties characteristic (read only)

Section 10.4 - Audio Input Control Service

In a further complexity, AICS allows an Audio Stream to have its Gain controlled automatically (traditionally known as Automatic Gain Control or AGC), or manually, which may either be by a Volume Control Client, or a local user control, with changed values exposed in the Audio Input State characteristic. When control is automatic, the Server does not support any changes made by the Client. The Server can expose whether or not the Client is allowed to change the mode from Automatic to Manual or vice versa through the Gain_Mode field of the Audio Input State characteristic, using the values shown in Table 10.8.

Gain_Mode	Value	Description
Manual Only	0	A Client may not change these settings
Automatic Only	1	
Manual	2	Manual, but a Client may change the Gain_Mode to Automatic
Automatic	3	Automatic, but a Client may change the Gain_Mode to Manual

Table 10.8 Meaning of the Gain_Mode field values in the AICS Audio Input State characteristic

Having sorted that out, we can move on to the Audio Input State and the Audio Input Control Point, which work much as we'd expect. The Audio Input State contains four fields, described in Table 10.9.

Name	Size (octets)
Gain_Setting	1
Mute	1
Gain_Mode	1
Change_Counter	1

Table 10.9 Fields for the Audio Input State characteristic

The Gain_Setting exposes the current value of Gain, in Gain_Setting_Units. Writing a value of 0 to Mute will not affect the current Gain_Setting value, so unmuting it by writing 1 to Mute will return the gain to the previous value in Gain_Setting. If the Server is in Automatic Gain Mode, it will ignore anything written to the Gain_Setting field.

Mute says whether the Audio Stream is muted (value = 0) or unmuted (value = 1). AICS gives us an additional option, where the value of 2 indicates that a Mute function is disabled. The final audio stream can still be muted using the VCS Mute, but by exercising this option, the individual input stream cannot be muted separately.

The Gain_Mode and the Change_Counter behave exactly as they do for VCS and VOCS. Again, it's an independent instantiation for each AICS instance.

Setting the AICS parameters is achieved by writing to the Audio Input State Control Point characteristic, with one of the five opcodes listed in Table 10.10.

If you are only implementing AGC and Mute functionality in your Acceptor, then there is no need to include an AICS instance for an audio stream, as unless you want to expose a friendly name to an app, everything else can be performed within VCS.

Opcode	Opcode Name	Description
0x01	Set Gain Setting	Sets the gain in increments of Gain_Setting_Units x 0.1dB
0x02	Unmute	Unmutes (unless mute is disabled)
0x03	Mute	Mutes (unless mute is disabled)
0x04	Set Manual Gain Mode	Changes From Manual to Automatic Gain (if allowed)
0x05	Set Automatic Gain Mode	Changes From Automatic to Manual Gain (if allowed)

Table 10.10 Opcodes for the Audio Input Control Point characteristic

These all do pretty much what it says in the name, although as we’ve already discovered, there are quite a number of caveats where the Server can ignore them, especially if it is not prepared to give up control of the Gain and Mute functionality.

The Set Gain Setting opcode (0x01) takes the form shown in Table 10.11.

Parameter	Size (Octets)	Value
Opcode	1	0x01
Change_Counter	1	0 to 255
Gain_Setting	1	-128 to 127

Table 10.11 Format for the Audio Input Control Point characteristic Set Gain Setting procedure

All of the other procedures use the same, shorter format for their parameters, shown in Table 10.12.

Parameter	Size (Octets)	Value
Opcode	1	0x02 = Unmute 0x03 = Mute 0x04 = Set Manual Gain Mode 0x05 = Set Automatic Gain Mode
Change_Counter	1	0 to 255

Table 10.12 Format for opcodes 0x02 - 0x05 for Audio Input Control Point characteristic

Section 10.5 - Putting the volume controls together

As with VOCS, there are a few additional characteristics which can help with a user interface.

The Audio Input Type is used to identify the Audio Input Stream with a read-only text description, which can be used for a user interface. Typical values (in English) are Microphone, Ambient, Bluetooth, etc. Values are defined in Section 6.12.2 of Assigned Numbers. These are set by the manufacturer.

The Audio Input Status exposes the current status of each AICS Audio Stream. If the value is set to 0, the Audio Stream is inactive; if it is set to 1, the Audio Stream is active.

The Audio Input Description allows for a more detailed description of an audio input, which is useful where there are multiple inputs of the same type, such as multiple Bluetooth or HDMI inputs. It may be set by the manufacturer or optionally made writable to allow a user to update it.

10.5 Putting the volume controls together

The opening diagram showed how these three services work together. Figure 10.2 shows a typical implementation for a pair of stereo headphones, which support both a Bluetooth connection and a local microphone.

The volume for each ear is set by combining the single value in VCS, with the Audio Location specific value from the VOCS instantiation for each ear. Either of the incoming audio streams can be individually muted or have their gain controlled (if the Server allows it), with VCS providing a global mute function.

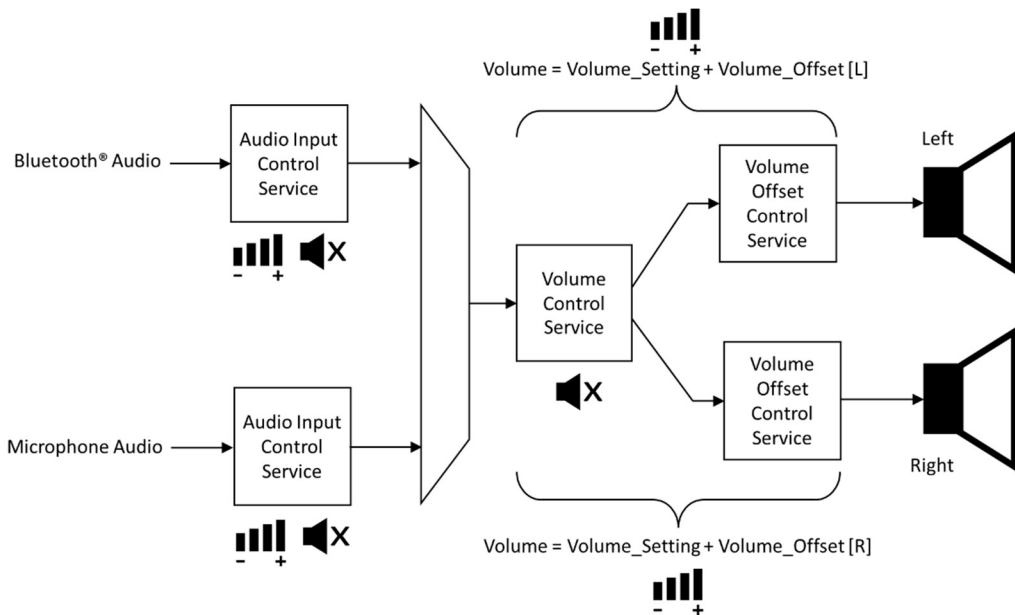


Figure 10.2 Typical implementation for a pair of headphones

Figure 10.3 shows a similar approach for the left hearing aid of a stereo pair. Because only one channel is being rendered, there is only one instance of VOCS, serving the left hearing aid. The other hearing aid, in the right ear, would look identical, but with a VOCS instance for the right Audio Location.

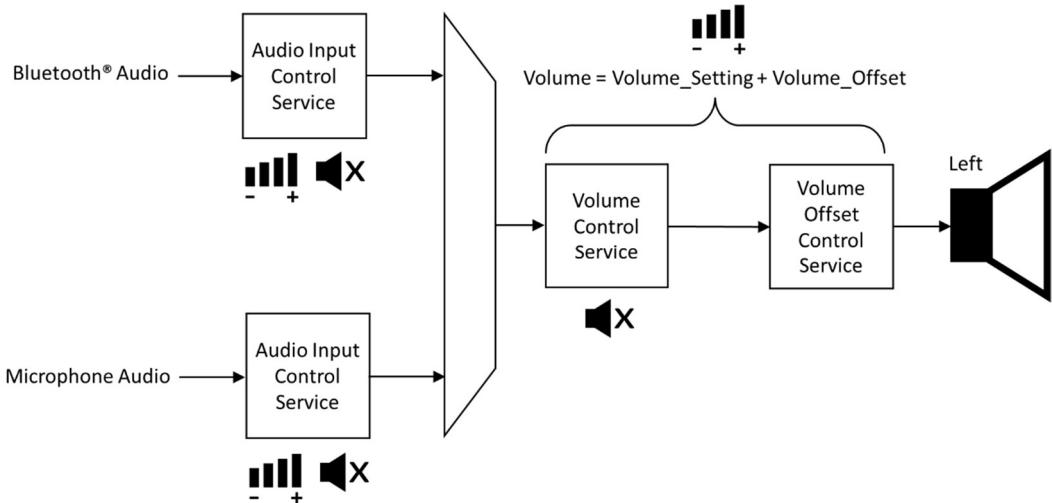


Figure 10.3 Typical implementation for a left hearing aid from a stereo pair

There is a practical consideration that designers need to appreciate, which is that different devices acting as remote volume controls may only support a subset of these services. For example, a simple volume control may only interact with the Volume Control Service, and not with AICS or VOCS, whereas a smartphone app could be able to support all three. This can lead to potential confusion. For example, if a phone app was previously used to mute the Bluetooth Audio input of Figure 10.3, a simple volume control that did not support AICS would not be able to unmute it. The Bluetooth specifications do not address this conflict, leaving it to individual implementations. One pragmatic solution would be to implement a local volume reset button to unmute all AICS and VCS instances whenever a user adjusts that local control.

10.6 Mixing and real world implementations

All of the diagrams in this section, as well as those in the LE Audio specifications show the AICS streams feeding into a mixing element. It's a simple hardware-centric visualization of how the various elements fit together.

It is certainly possible to implement volume control in that way, but real world implementations are generally far more complex, especially in hearing aids and earbuds. Figure 10.4 illustrates a more typical implementation, where the VCS element not only provides the overall gain, but also takes control of the operation of the various AICS streams.

Section 10.7 - Microphone control

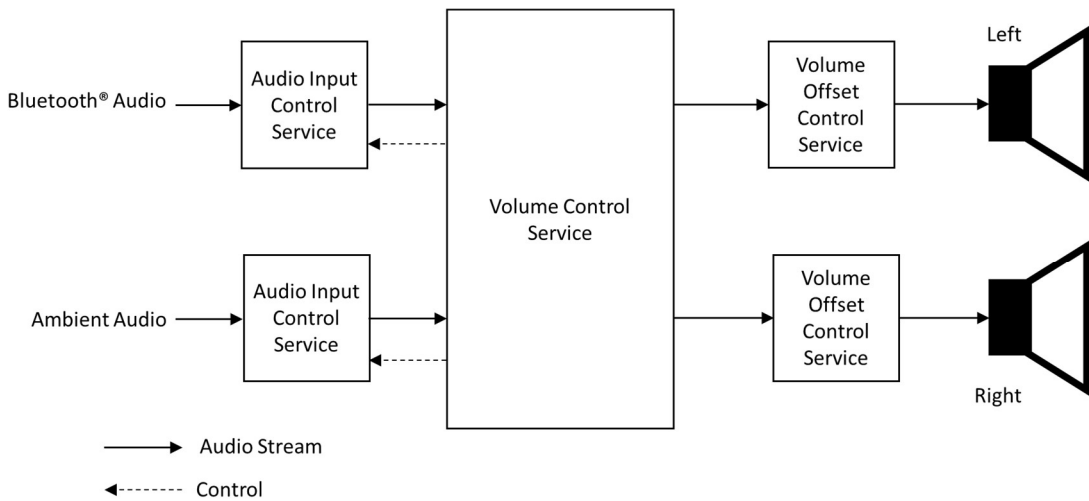


Figure 10.4 A real-world implementation of the volume control elements

The reason for this is that the relative gains of the different inputs often need to be managed based on the context, which may involve what level of transparency is required, and in the case of hearing aids, on the specific preset which has been chosen (See Section 11.1). It may also be affected by other audio processing which is applied, such as ANC. Even Figure 10.4 may be a simplification, as all of the elements are typically contained within a complex gain processing block.

Users are not aware of this level of detail, so it is important that product designers think carefully about the user experience, particularly where a variety of different volume controllers are used. They should also be aware that the local control of volume on earbuds and hearing aids may be limited by the lack of a user interface, but should never be allowed to get stuck in a state which cannot be easily reset by the user.

10.7 Microphone control

Although the examples above include microphones, they are all using the microphone as a local input, where the microphone stream is selected or mixed with other audio inputs to be rendered. However, microphones are also used as audio inputs which are captured and sent back to an Initiator. The Microphone Control Service and Profile (MICS and MICP) exist to provide control over Microphones which are used for Audio Sources.

The Microphone Control Service is probably the simplest service in all of the Bluetooth LE Audio specifications, comprising a single characteristic, which provides a device-wide mute for microphones. Its simplest usage is shown in Figure 10.5.

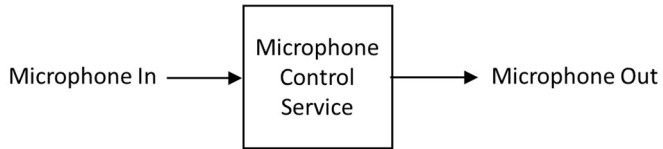


Figure 10.5 Simple usage of the Microphone Control Service

There is no corresponding Control Point characteristic. Instead, the Mute Characteristic can be written directly, as well as being read and notified. It has the same features as the Mute field in the AICS Audio State characteristic, namely:

Description	Value
Not Muted	0x00
Muted	0x01
Mute Disabled	0x02

Table 10.13 MICS Mute characteristic values

If control of microphone gain is required, MICS should be combined with an instance of AICS (Figure 10.6), although in this case MICS doesn't provide much advantage over just using AICS. However, most Clients would expect to use MICS to perform a microphone mute, hence the reason to implement MICS.

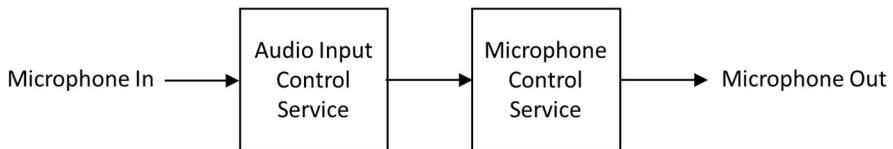


Figure 10.6 Combining the Microphone Service with an instance of the Audio Input Control Service

The combination of AICS and MICS makes more sense when multiple microphones exist, as MICS gives the benefit of a device-wide mute for the microphones, which is its real purpose (Figure 10.7).

Section 10.8 - Local volume control

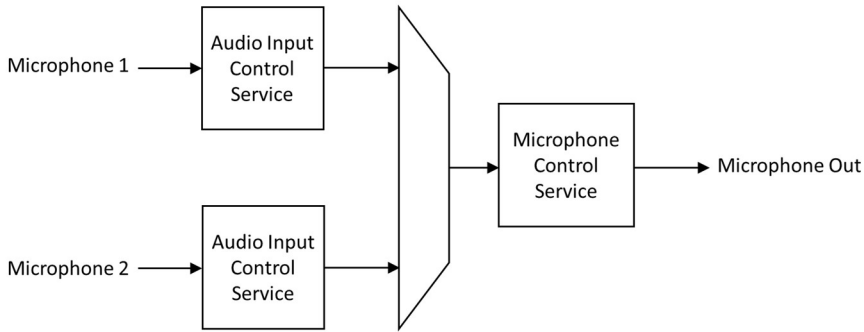


Figure 10.7 Use of MICS and AICS for multiple microphones

Finally, MICS can be used to provide a device-wide mute for multiple microphones as part of a volume control scheme, as shown in Figure 10.8. However, in most cases, multiple microphones in an Acceptor will be feeding directly into audio processing modules, so it's unlikely that external control of individual microphones would be a requirement. What Figure 10.8 does is demonstrate the flexibility of volume and microphone control services in Bluetooth LE Audio.

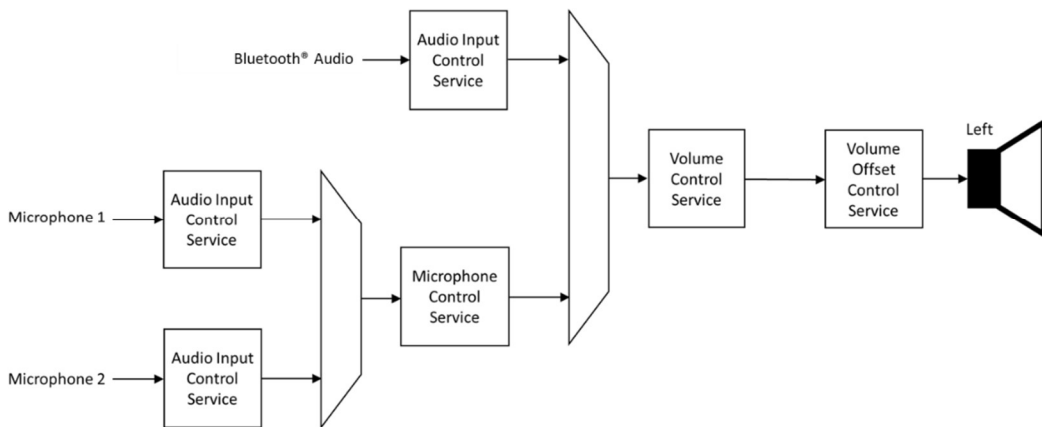


Figure 10.8 The combination of Microphone and Volume Control Services

As with AICS, designers should be aware of the impact on user experience if some of their remote controls do not act as Clients for all of the Acceptor's volume and microphone services.

10.8 Local volume control

Everything covered in this chapter has looked at how the volume at which audio is rendered at an Acceptor can be controlled over a Bluetooth connection. Most earbuds, headsets and hearing aids, as well as some speakers also permit local volume control, typically with knobs, buttons or gestures. Where these are implemented, any local change should be reflected in the appropriate characteristic and notified to all of the registered volume controllers.

Normally, this will be limited to the Volume State in VCS, as local controls are normally limited to overall volume.

Many earbuds and hearing aids contain a proprietary radio link between their left and right devices which will transfer a volume change to the other device. In this case, both will send notifications on their new values.

When a local control results in a change in value of the Volume_Setting or Mute fields of the Volume State characteristic, it should also increment the value of the Change_Counter, to prevent any confusion from occurring when Volume Controllers next write to it.

10.9 A codicil on terminology and multiple Volume Controllers

Throughout this book I've been using the terms Initiator, Acceptor and Commander to describe the three main types of device in the Bluetooth LE Audio ecosystem. As I stated in Chapter 3, these terms are defined as roles in CAP, so purists would probably object to my conflating them with devices. I still feel that conflation leads to a clearer understanding of how everything fits together.

Potential for confusion exists in the way a Commander (as a device) interacts with Initiators and Acceptors. As a role, a Commander can be collocated⁷¹ with an Initiator, but as a device, its interactions with an Initiator and each of its Acceptors can be confusing. Although all of these interactions are covered by CAP procedures, they are independent. Figure 10.9 illustrates some of the profiles and services described in this and the previous chapter in a simple case of an Initiator, Acceptor and independent Commander.

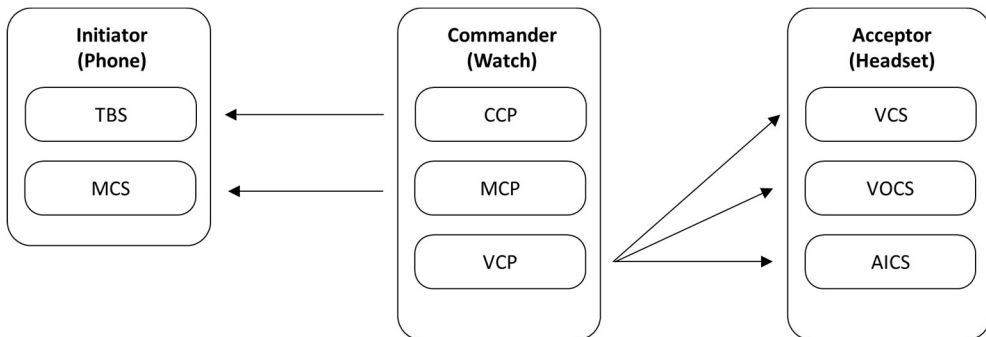


Figure 10.9 An example of Profile and Server relationships for volume and control

In this case, three profiles are implemented in the Commander. Two of them – Call Control

⁷¹ The spelling (or misspelling) of colocation and collocation can add further confusion. These are not alternative spellings, but totally different words. Colocation, with one “l” means “in the same place”, deriving from the Latin “locare”. In contrast, collocation, with two “l”s means “working together”, coming from the Latin “collegium”. In many cases, both may be appropriate at the same time.

Section 10.9 - A codicil on terminology and multiple Volume Controllers

and Media Control act on the complementary services in the Initiator, whereas Volume Control operates on the VCS, VOCS and AICS instances in the Acceptor. For the grammatically inclined, the profiles in the Commander and services in the Initiator are collocated; the services in the Acceptor are collocated.

Although that make seem complicated, the reality will quickly become even more so. In the example above, it's likely that both the phone and the watch will contain instances of VCP. They may not have the same functionality. Whereas the instance on the phone (which may be a native application, or included in an audio or headset application is likely to support VCS, VOCS and AICS, the VCP instance on the watch may only support VCS.

Figure 1.1Figure 10.10 shows the Client Server relationships for this arrangement, adding in the local volume control on the headset. The user has three different ways of adjusting the overall volume of what they are hearing – locally on the headset control, remotely on the watch, or on the phone. The VCP instances on the watch and phone should maintain details of the current server settings by receiving notifications of any change in the Volume State characteristic, regardless of which device caused that change. This means that any applications on the device can always display the current state of the headset volume.

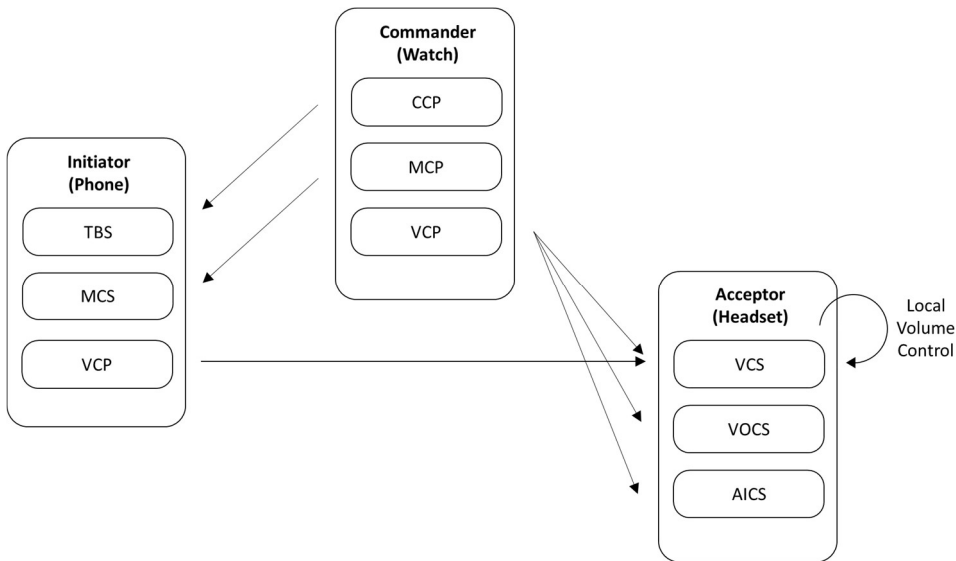


Figure 10.10 Client Server relationships for multiple Volume Controllers

If the headset has a sufficient UI, there's no reason why it can't also implement instances of CCP and MCS, allowing a user to answer an incoming phone call or move to the next track on their music app. The state of each feature is held in one place, and multiple clients can operate on it.

That brings us to the end of the specifications within the Generic Audio Framework. We now need to move on to the top level profiles, and the Broadcast Audio URI specification.

Chapter 11. Top level Bluetooth® LE Audio profiles and the Broadcast Audio URI

Having covered everything in the Generic Audio Framework, we now come to the top level profiles of Bluetooth LE Audio. Although they're still called profiles, in most cases they're a lot simpler than the underlying profiles we've discussed in the Generic Audio Framework, or the Bluetooth Classic Audio profiles. Instead of defining procedures, they generally confine themselves to configuration, specifying new roles which mandate a combination of optional features, and adding QoS requirements beyond those of BAP. In doing so they raise the bar for implementations by defining feature combinations to meet commonly experienced use cases.

In this chapter we will look at what these profiles contain. As they rely on features which are already defined in the GAF specifications, they do not have separate complementary service specifications. HAP – the Hearing Access Profile⁷² is the exception, as the corresponding Hearing Access Service introduces a new Presets characteristic for Hearing Aids. TMAP – the Telephony Media and Audio Profile has a nominal TMAP service, which is included in the TMAP specification. GMAP – the Gaming Audio Profile, follows the same principle of including GMAS. Both of these services confine themselves to exposing which roles the Server supports. PBP - the Public Broadcast Profile has no service. That's an anomaly that is inherent with a broadcast application which does not expect a connection between Initiator and Acceptor. The lack of connection implies no possibility of a Client-Server relationship, hence no opportunity for a Service specification.

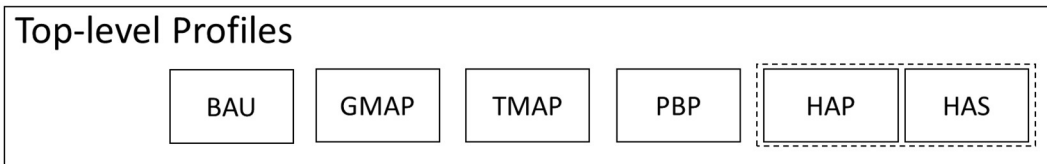


Figure 11.1 Current Top level Bluetooth LE Audio profiles

One other Bluetooth LE Audio specification has been published, which is the Bluetooth Audio URI specification (BAU). This is also different, as it defines a Universal Resource Identifier (URI) scheme to expose broadcast advertising information over non-Bluetooth transports, such as QR codes or Near Field Communication (NFC). It is intended to simplify the user experience of synchronizing to broadcast streams.

⁷² The Hearing Access Profile and Service were previously called the Hearing Aid Profile and Service, but the names were changed as the term “Hearing Aid” has a specific, regulated medical meaning in some countries, so a declaration that a product complied with a Hearing Aid specification could cause confusion.

As the whole of the Bluetooth LE Audio development was started by the hearing aid industry, it seems only fair to start with HAP and HAS, which define requirements for products which are intended for use in the hearing aid ecosystem.

11.1 HAPS - the Hearing Access Profile and Service

The Hearing Access Profile and Service have been designed to meet the requirements of devices which are used in the Hearing Aid ecosystem, which covers hearing aids, products which supply Audio Streams to hearing aids and accessories which are used to control them.

Hearing Aids are different to earbuds and other Acceptors because they are always on, capturing and processing ambient sound to assist their wearers. That means that all of the use cases driving the Hearing Access profile envisage Bluetooth LE Audio as an additional audio stream to the ambient one. This makes them unusual, as Bluetooth connectivity is not the de facto reason for using them. They also differ in that everyone who wears them has hearing loss, so they have a different balance in requirements between audio quality (and hence QoS settings) and battery life.

Taking your hearing aid out to recharge it during the day is a far more significant action than it is for an earbud wearer, as you may not be able to hear during that time. Pushing up audio quality increases the power consumption, so the Hearing Access profile imposes no additional QoS requirements over the settings mandated by BAP, typically using the 16_2 QoS settings for voice (16kHz sampling rate, 7 kHz bandwidth) and the 24_2 QoS settings for music (24 kHz sampling rate, 11 kHz bandwidth). As many hearing aids do not occlude the ear, ambient sounds can be heard in addition to the Bluetooth stream. For this reason, hearing aids generally prefer to use the Low Latency QoS settings to avoid introducing echo between the ambient and transmitted sound.

The HAP specification describes the physical device configuration of products recognised as hearing aids. The profile defines four different configurations of Acceptor(s), all of which are included in the general term “hearing aid” throughout the document. These are:

- A single hearing aid, which renders a single Bluetooth LE Audio Stream,
- A pair of hearing aids (also called a Binaural Hearing Aid Set) which are the members of a Coordinated Set, supporting individual left and right Audio Channels for each ear,
- A single hearing aid that receives separate left and right Audio Streams and combines the decoded audio into a single Bluetooth LE Audio Channel, and
- A hearing aid that uses a single Bluetooth link carrying separate left and right audio channels which are rendered at the appropriate ear. These are called Banded Hearing Aids, where there is a wired connection between the hearing aid device in each ear and the Bluetooth transceiver, which is typically in a neckband or an over-the-head band.

Section 11.1 - HAPS - the Hearing Access Profile and Service

The Hearing Access Profile defines four different roles for the hearing aid ecosystem:

- HA (Hearing Aid), which is any one of the four types of hearing aid described above.
- HAUC (Hearing Aid Unicast Client), which is an Initiator which can establish a unicast Audio Stream with one or more hearing aids. The unicast Audio Streams can be unidirectional or bidirectional.
- HABS (Hearing Aid Broadcast Sender), which is an Initiator transmitting broadcast Audio Streams, and
- HARC (Hearing Aid Remote Controller), which is a device that controls volume levels, mute states and hearing aid presets (which we'll come to in a minute).

Note that HAP does not include a Broadcast Assistant, as that functionality is fully described in BAP and BASS and does not require any new role.

The bulk of the Hearing Access Profile lists mandatory combinations of BAP and CAP features which are required for different product applications. Whilst many may seem obvious, they ensure that any products claiming support for the profile will contain the same features and be interoperable. This is essentially the point of Bluetooth LE Audio top level profiles – ensuring that interoperability is assured for specific use cases by clearly specifying which underlying profiles, services and features must be present.

There are limitations. Every hearing aid must be capable of receiving both Unicast and Broadcast Audio Streams that comply with the BAP mandated requirements, i.e. the 16 and 24kHz sampled configurations with 10ms frames. They do not need to receive audio streams which have been encoded with other optional QoS settings. A hearing aid must support the volume controller role, accepting commands from any device which supports the HARC role, which could be a stand-alone remote control, an application on a phone, or physical volume button on an Initiator. For the first time, this means that hearing aid accessories will be globally interoperable with any make of hearing aid. This allows a consistent user experience where a device can be used for volume control across unicast and broadcast use cases.

All hearing aids need to support the «Ringtone», «Conversational» and «Media» Context Types, which means that they will all support incoming phone calls. However, they do not need to support Call Control or return a voice stream when in a phone call. That's a practical decision, as many hearing aids locate their microphones for best pickup of sound around the user, rather than the wearer's voice, so it will often be better for a user to use the phone's microphone when in a call. These are limitations which manufacturers will need to convey in their product marketing.

HAP and HAS introduce a new concept, which is support for Presets. Presets are proprietary audio processing configurations which hearing aid manufacturers include to optimise the sound fed to the ear. Typically, they adjust the processing to cope with different environments,

such as restaurants, shops, office, home, etc. HAP and HAS don't attempt to standardise these settings, but provide a numbering scheme which manufacturers can map to their specific implementation. Users can then select a specific preset by its number, or cycle through them. It includes the ability to add Friendly Names, so that an application can display the current preset and other available presets. It also supports dynamic presets, where the availability of a preset may change depending on the status of the hearing aid. For example, a preset that was optimised for reception of a telecoil signal would not be available when there is no telecoil loop available. Presets are currently a feature which are specific to hearing aids. For more information on them, refer to the HAPS specifications.

Within the preset feature of HAS, there is an interesting option, which is likely to expand into other profiles. This is the ability to use a non-Bluetooth radio to relay a command from one hearing aid to the other without the need for a Commander to institute a notification-based procedure to relay the change to other set members. This is the Synchronized Locally operation, which informs any Commander present that a preset command sent to one hearing aid can be locally relayed to the other member of the set, which is described in Sections 3.2.2.8 – 3.2.2.8 of HAS. It means that a Commander only needs to write to the Hearing Aid Preset Control Point characteristic of one set member.

The other enhancement in HAP is the option to include the Immediate Alert Service [HAP 3.5.3]. This allows a device which does not support an Audio Stream to provide an alert to the Hearing Aid, which it can render as an alert to the user. No specific use cases are defined, but it is suggested that manufacturers might want to include this capability in products such as doorbells or microwave ovens to help inform hearing aid wearers of something they need to respond to.

The aim of the HAP requirements is to support all of the use cases envisaged for hearing aids, which are illustrated in Figure 11.2.

Section 11.1 - HAPS - the Hearing Access Profile and Service

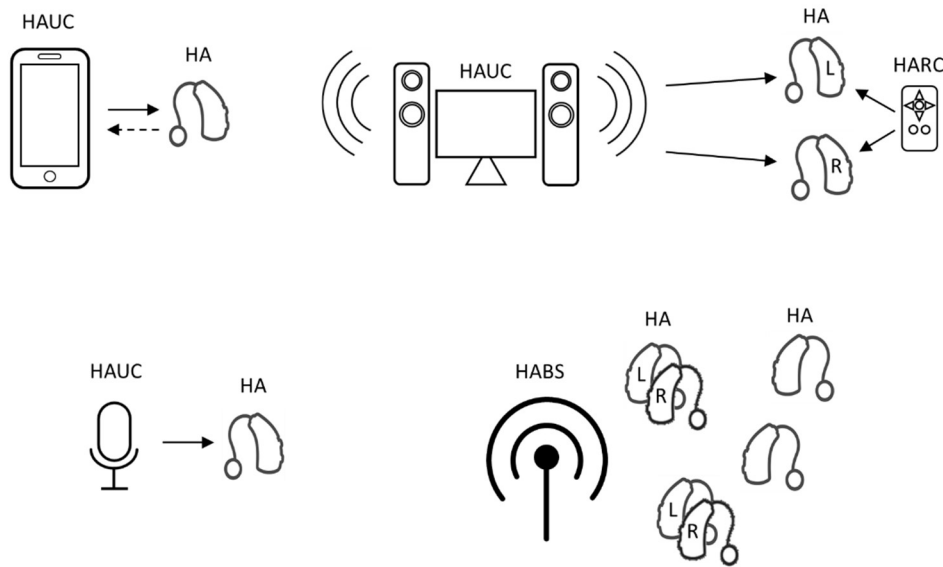


Figure 11.2 The four main use cases for HAP, showing the roles used in each

As many of the hearing aid use cases will involve the user hearing the ambient sound as well as the received Bluetooth LE Audio Stream, it is expected that devices will favour the use of Low Latency QoS modes. To help ensure low latency, there is a requirement that all devices in the HA role shall support a Presentation Delay value of 20ms for receive and transmit for Low Latency QoS modes. It means that Hearing Aids must be able to support any value for Presentation Delay between 20.000 and 40.000 msecs, as it builds on the 40ms requirement of BAP.

A Broadcast Transmitter may use another approach to minimise overall latency, which is to include the Broadcast Audio Immediate Rendering Flag Metadata LTV (BAIRF) in either its Public Broadcast Announcement, or BASE subgroup. This indicates that Broadcast Receivers can minimise the overall latency by rendering the signal as quickly as they are able, ignoring the value of Presentation Delay in the BASE. If two hearing aids are members of a Coordinated Set, they need some mechanism to ensure that they both use the same value. This is normally accomplished by a manufacturer setting a default BAIRF value in their products. Different sets of hearing aids are likely to implement different values for accommodating BAIRF, which means that wearers will hear their rendered audio at slightly different times, with the worst case being determined by the BASE value of Presentation Delay, which will be used by devices which do not support BAIRF. Hence it is a feature which helps optimise more advanced devices.

A final nuance in the Hearing Access Profile is a set of requirements in Section 4.1, which dictate Link Layer settings when the HAUC is using a 7.5ms Isochronous Interval and the hearing aid does not support the reception of 7.5ms LC3 codec frames. It is expected that this will be an edge case in situations where a hearing aid has a minimal LC3 implementation

(as support for 7.5ms codec frames is not mandated) and where an Initiator is forced to use a 7.5ms interval because of timing limitations of its other peripheral devices, which cannot accommodate the preferred 10ms interval.

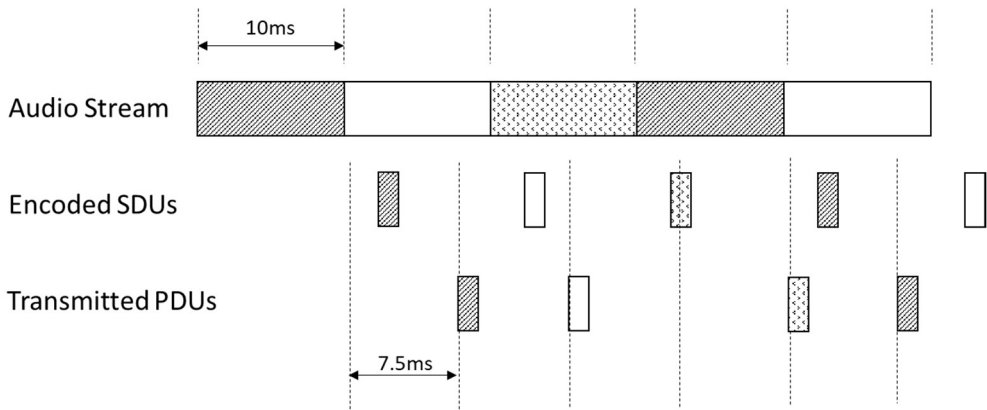


Figure 11.3 The effect of combining 10ms capture with 7.5ms transmission

The problem is illustrated in Figure 11.3, which shows an audio stream being sampled at 10ms, resulting in encoded SDUs being available every 10ms. At the bottom, the 7.5ms transmission of PDUs shows the misalignment and the need to allocate the encoded SDUs to the 7.5ms slots.

In this case, the scheduler should ensure the selection of the specified parameters for ISOAL to avoid segmentation of SDUs and a potential increase in the frame loss rate. It is anticipated that Initiators will move to the optimum 10ms transport interval as Bluetooth LE Audio becomes common, hence this is a short-term fix. Bluetooth® Core 5.3 introduced Enhanced ISOAL, which is a better solution. This section remains in HAP to provide backwards compatibility.

All HAUCs, HAs and HABs must support the 2M PHY. Although its use is not mandated, it is highly recommended to conserve airtime, and hence battery life.

11.2 TMAP – The Telephony and Media Audio Profile

Like HAP, the TMAP profile is predominantly a list of additional requirements beyond those mandated in BAP and CAP to emulate the use cases of the HFP and A2DP profiles. TMAP does not introduce any new behaviour. It incorporates a minimal TMAP section within TMAP, with a characteristic which exposes the roles which an Acceptor supports.

Because it is bundling both telephony and media applications into one document, TMAP feels like a portmanteau profile, where implementers have the ability to pick and choose what they support. This can lead to some oddities, as many of its features are optional. In theory, depending on what you pick, it is possible to make a TMAP compliant device which supports telephone calls, but does not support music and vice versa. That's a logical consequence of

Section 11.2 - TMAP – The Telephony and Media Audio Profile

bundling so many different use cases together. It makes sense that a soundbar or speaker doesn't support telephony, but that means that headphones don't need to support telephony either. Instead, it is up to the implementer to choose the features and roles which are appropriate and let market Darwinism take care of any inauspicious choices.

TMAP requires support for the Volume Controller role for all unicast Initiators (CG and UMS), and mandates support for the Volume Renderer role when the Acceptor is acting as an Audio Sink.

To try and prevent any surprises, the TMAP element of TMAP includes a TMAP Role characteristic [TMAP 4.7], which exposes the specific roles which a device supports. This allows Acceptors to determine the Roles that an Initiator supports, which is required for some use cases. Which brings us to the Roles. TMAP does not define any new Roles for a Commander, but defines three pairs of Roles for an Initiator and an Acceptor.

11.2.1 Telephony Roles – Call Gateway and Call Terminal

For telephony, TMAP defines a Call Gateway (CG) and Call Terminal (CT) Role. The Call Gateway is the device which connects to the telephony network, such as a phone, tablet or PC, and is an Initiator. A Call Terminal is typically a headset, but could also be an extension speaker, or a microphone for a conference phone. Some common configurations are shown in Figure 11.4

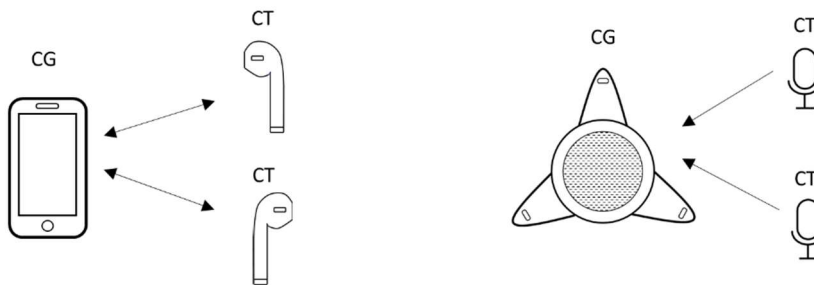


Figure 11.4 Typical configurations for the CG and CT roles

Devices supporting the CG and CT Roles must support the higher codec settings of 32 kHz sampling, at both 7.5ms and 10ms frame rates (32_1 and 32_2 from BAP), with the Low Latency settings. These correspond to the same audio quality as the Superwideband EVS speech codec that the 3GPP specifies for 5G phones, ensuring no loss of quality from local microphone to remote headset and vice versa.

A CG must support the CCP Server role, as the CG is where the state of the call is stored. TMAP does not mandate any further features above those mandated in Table 3.1 of TBS.

11.2.2 Media Player Roles – Unicast Media Sender and Unicast Media Receiver

Media player use cases employ the Unicast Media Sender (UMS) and Unicast Media Receiver (UMR) Roles. The Unicast Media Sender is the Initiator, which is the audio source; the Unicast Media Receiver is an Acceptor. Typical configurations are shown in Figure 11.5.

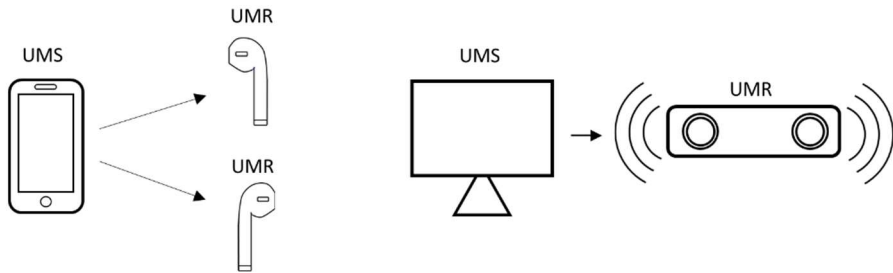


Figure 11.5 Typical TMAP Unicast Media applications

For Unicast Media, TMAP pushes up the audio quality, requiring a Unicast Media Receiver to support all six of the 48kHz sampling codec configurations, from 48_1 to 48_6. Unicast Media Sources must support the 48_2 codec setting and at least one of the 48_4 or 48-6 settings. All TMAP roles must support the 2M PHY, which will almost certainly be necessary to find enough airtime for these configurations. Although support for these QoS configurations is required, it is up to the application to decide how it configures the ASEs. It can decide to use lower values. For UMS, the 32 kHz sampling rates remain optional, although it is unlikely that an Acceptor would not support them. A user is unlikely to hear the difference between 32 and 48kHz, so the choice of setting by the application may be a pragmatic decision based on device-wide airtime constraints resulting from the wireless activity of other applications. Initiators should be aware of the impact of these settings on their Acceptors, particularly if they are hearing aids. Supporting the most intensive QoS settings may not provide the optimal user experience.

TMAP does not mandate the support of any Context Type other than «Unspecified». If a Unicast Media Receiver wants to reject any attempt to establish a stream from a Call Gateway, it should support «Ringtone», but set it to “not available”.

TMAP increases the requirement for media control by mandating support for Play and Pause opcodes. A UMS must also support the Media Control Point characteristic.

Section 11.2 - TMAP – The Telephony and Media Audio Profile

11.2.3 Broadcast Media Roles – Broadcast Media Sender and Broadcast Media Receiver

The final two sets of Roles in TMAP are the Broadcast Media Sender (BMS) and Broadcast Media Receiver (BMS) roles. Unsurprisingly, these are designed for broadcast applications. Within TMAP they are generally envisaged to be personal applications, where higher audio quality is required, but may also be used in public broadcast applications, such as cinemas. Typical use cases are shown in Figure 11.6.

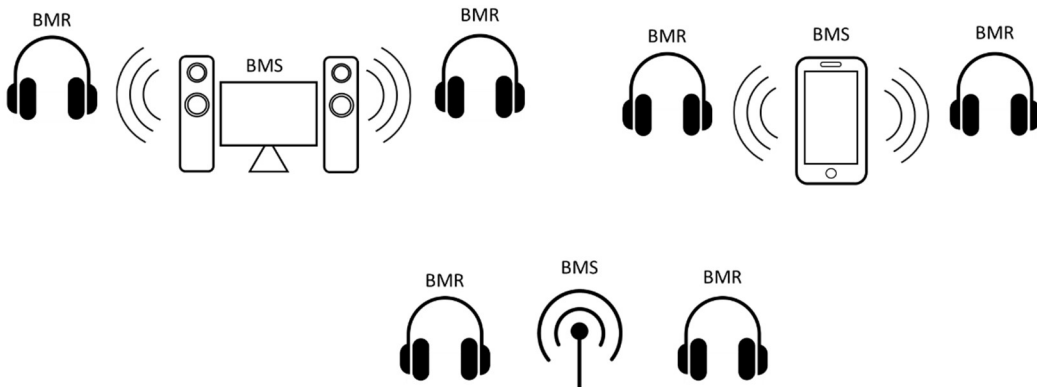


Figure 11.6 Typical use cases for the Broadcast Media Sender and Receiver roles

TMAP adds very few requirements on top of BAP and CAP for the broadcast Roles. It mandates support for higher quality QoS settings, requiring support for all of the Low Latency and High Reliability 48 kHz QoS modes defined in table 6.1 of BAP (48_1_1 to 48_6_1 and 48_1_2 to 48_6_2) for a Broadcast Media Receiver and both 48_1 and 48_2 QoS configurations for a Broadcast Media Sender. It also mandates that a Broadcast Media sender must support at least one of the 48_3 or 48_5 (7.5ms frame) codec configurations and one of the 48_4 or 48_6 (10ms frame) codec configurations.

TMAP requires that Broadcast Media Receivers support a Presentation Delay value of 20ms within their range of Presentation Delays for both Low Latency and High Reliability QoS modes. Added to the BAP requirements, this means that Broadcast Media Receivers must support any value of Presentation Delay between 20.000 and 40.000msecs. There is currently no similar requirement on the unicast roles.

TMAP also increases the mandatory support for broadcast Audio Configurations, requiring that a Broadcast Receiver can accept any of the broadcast Audio Configurations defined in Table 4.24 of BAP. These are shown in Table 11.1. The BMS requirements are unchanged from BAP.


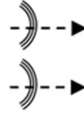

Audio Configuration	Stream Direction (Initiator – Acceptor)	BMS	BMR
12		M	M
13		M	M
14		O	M

Table 11.1 Audio Configuration requirements for Broadcast Media Roles

11.3 GMAP – the Gaming Audio Profile

The Gaming Audio profile has been written to define specifications for using Bluetooth LE Audio in gaming and low latency applications. As with HAP and TMAP, it includes roles covering both unicast and broadcast audio streams. It concentrates on low latency sound and unlike HAP and TMAP, it introduces new QoS configurations for the LC3 codec. GMAP does not include any new procedures, but builds on those from CAP. It does include recommendations for Link Layer settings.

GMAP mandates implementing the Volume Controller role for the Unicast Game Gateway (UGG), and the Volume Renderer role for all Acceptors acting as Audio Sinks. It does not require support for any telephony or media control roles. All GMAP compliant devices need to support the use of the 2Mbps PHY.

11.3.1 Unicast Roles – Unicast Game Gateway and Terminal

The unicast roles within GMAP look very similar to the Call Gateway and Call Terminal roles of TMAP. The Unicast Game Gateway (UGG) is implemented on the Initiator and the Unicast Game Terminal (UGT) on the Acceptor(s), as shown in Figure 11.7.

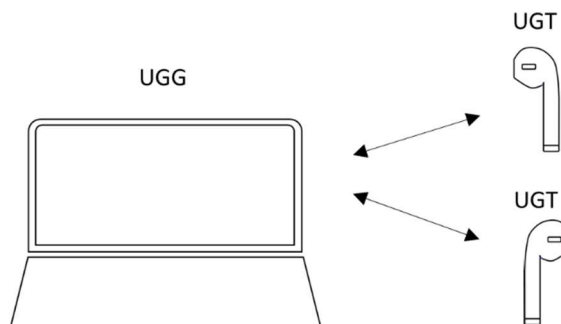


Figure 11.7 Typical configurations for the UGG and UGT roles

The difference is in the detail of their configuration. Each outgoing stream from the UGG must be capable of supporting a high quality, 48kHz sampled LC3 stream, which may

Section 11.3 - GMAP – the Gaming Audio Profile

optionally be multiplexed. The return streams, which use a bidirectional CIS, must be capable of supporting 32kHz sampled LC3 streams and may optionally support 48kHz.

To obtain the desired audio quality and latency, GMAP introduces a new set of QoS configurations, optimising the BAP configurations which are used by other top level profiles. These use the same Codec Capabilities nomenclature as BAP, but replace the suffix with either “_gs” or “_gr”, indicating whether they are applied to the Terminal to Gateway direction. The nomenclature is confusing, as it refers to the Acceptor. “_gs” means Sent from the Unicast Gaming Terminal; “_gr” means Received by the Unicast Gaming Terminal. Although the Acceptor is the datum for the direction, the choice of QoS setting is always made by the Initiator. The key parameters for these new configurations are shown in Table 11.2 and Table 11.3.

Set Name	SDU Interval (ms)	Bitrate (kbps)	RTN	Max Transport Latency (ms)	Presentation Delay
16_1_gs	7.5	32	1	15	60
16_2_gs	10	32	1	20	60
32_1_gs	7.5	64	1	15	60
32_2_gs	10	64	1	20	60
48_1_gs	7.5	80	1	15	60
48_2_gs	10	80	1	20	60

Table 11.2 Additional GMAP QoS configurations for streams sent from the UGT to the UGG

Set Name	SDU Interval (ms)	Bitrate (kbps)	RTN	Max Transport Latency (ms)	Presentation Delay
32_1_gr	7.5	64	1	15	10
32_2_gr	10	64	1	20	10
48_1_gr	7.5	80	1	15	10
48_2_gr	10	80	1	20	10
48_3_gr	7.5	96	1	15	10
48_4_gr	10	96	1	20	10

Table 11.3 Additional GMAP QoS configurations for streams sent from the UGG to the UGT (received by the Terminal)

Comparing these with the standard configurations in BAP, the Maximum Transport Latency is significantly reduced, largely by reducing the number of retransmissions.

The range of Presentation Delays which a UGG supports is also widened. A Sink ASE must support a range of values from 10ms to 40ms, and a Source ASE must support values from

40ms to 60ms. The Gateway to Terminal value of 10ms, is currently the lowest value of any top level Bluetooth LE Audio specification. Using these new values, it should be possible to guarantee very low latencies for audio streams directed to Acceptors.

To allow gaming applications to further optimise latency, GMAP defines four different Levels of Quality of Service, which allow an application to select a QoS option. These are described in Table 11.4:

Level	Definition	Description
A	Lowest latency	The lowest latency possible. This might result in a poorer user experience if interference causes lost packets.
B	Low latency	More retransmissions than Level A (Lowest latency) which helps robustness at the cost of slightly higher latency.
C	Balanced	Increased robustness, which also increases the latency.
D	High reliability	The best robustness within GMAP while meeting the latency limits.

Table 11.4 The GMAP QoS levels

The GMAP specification contains a large amount of information about how the different QoS configurations are applied to different Audio Configurations as well as the mandatory and optional requirements for Initiators and Acceptors. It also provides example figures for the anticipated total system latency which can be achieved for the Gateway to Terminal direction, which ranges from 26.4ms to 42.4ms. Appendix A of GMAP explains how these figures are calculated and is an extremely useful resource to understand the build-up of latency elements in a Bluetooth LE Audio application.

GMAP introduces asymmetry to help achieve its target latencies. Figure 11.8 shows an example where a laptop is streaming to a pair of earbuds. The left earbud is receiving a 48 kHz, game audio with game chat stream and returning a 32kHz game chat stream using the bidirectional CIS. The right earbud is just receiving a left channel 48 kHz, game audio stream. The latencies are typical for a Level B GMAP QoS implementation.

Section 11.4 - Broadcast Gaming Roles – BGS and BGR

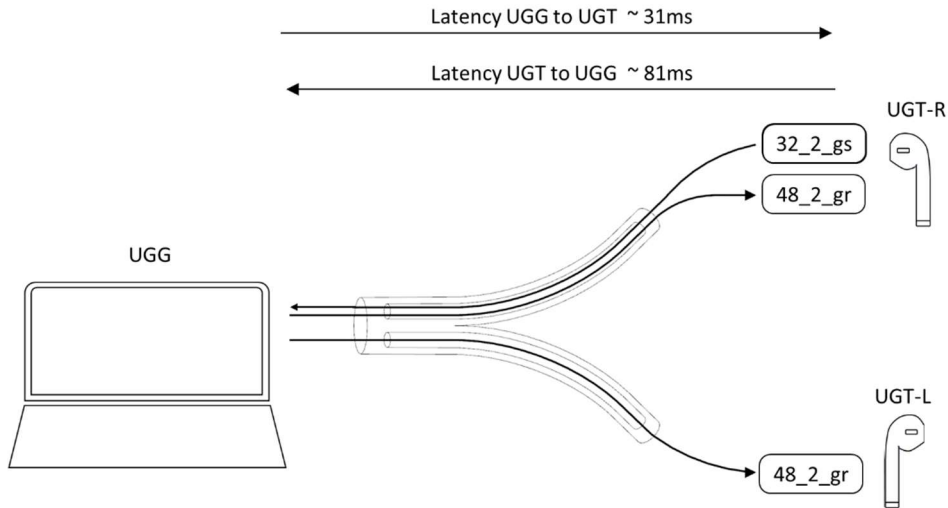


Figure 11.8 An example of the asymmetry of QoS settings for a unicast GMAP application.

For most gaming applications, the latency of the game chat (which originates from the UGT) is not considered as important as the game audio coming from the UGG. Within GMAP, the Initiator can use the feature bits described in Section 11.6 to determine Acceptor capabilities and optimise the configuration.

A UGG can transmit individual left and right audio streams in each CIS, or the same multiplexed Left + Right stream to both. Implementers should bear in mind that the larger, multiplexed packets are more susceptible to interference, which may affect robustness when there is only one retransmission scheduled. This is unlikely to be an issue in a home environment, but may be a consideration in noisier wireless locations.

11.4 Broadcast Gaming Roles – BGS and BGR

GMAP supports broadcast in a similar manner to TMAP, with two new roles – a Broadcast Game Sender (BGS) and a Broadcast Game Receiver (BGR), shown in Figure 11.9.

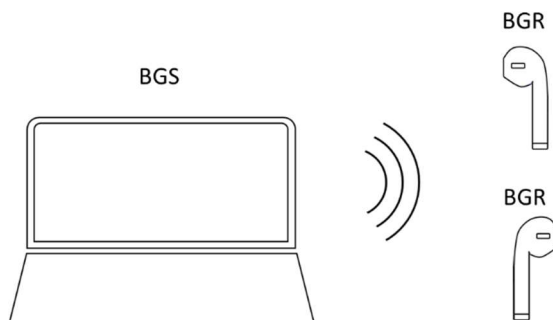


Figure 11.9 An example of the Broadcast gaming roles

As with the unicast roles, GMAP introduces new audio stream configurations which specify higher sampling rates and different values of Presentation Delay, as shown in Table 11.5. In

this case the configurations apply to broadcast – there is no return path. These configurations are characterised with the suffix *_g* applied to the Codec Capability.

Set Name	SDU Interval (ms)	Bitrate (kbps)	RTN	Max Transport Latency (ms)	Presentation Delay
48_1_g	7.5	80	1	8	10
48_2_g	10	80	1	10	10
48_3_g	7.5	96	1	8	10
48_4_g	10	96	1	10	10

Table 11.5 Additional audio stream configurations for Gaming Audio broadcast roles

The Maximum Transport Latency is limited to one Isochronous Interval, corresponding to the SDU interval, with only one retransmission. Broadcast Game Receivers must support all of these new audio stream configurations. The limitation on the number of retransmissions implies a lower level of robustness, particularly if multiplexed Left and Right packets are being used. However, most gaming application are likely to be used by personal players who are close to the device acting as the Broadcast Game Sender, which mitigates this issue. As with unicast gaming, four different levels are defined, resulting in a total system delay of around 30ms for a 48_2_g stream.

11.5 GMAP synchronization

GMAP is the first top level specification to impose synchronization requirements for UGT and BGR roles, mandating that a pair of Acceptors must be able to render audio within $\pm 100\mu\text{s}$ of each other, with a maximum jitter of $\pm 25\mu\text{s}$.

11.6 GMAP features and GMAS

GMAP defines some new features which can be applied to both unicast and broadcast configurations, which are described in Table 11.6:

Feature	Description
96 kbps Sink / Source	Support for audio stream configurations of 48_3 and 48_4, corresponding to a bit rate of 96kbps ¹ .
80 kbps Sink / Source	Support for audio stream configurations of 48_1 and 48_2, corresponding to a bit rate of 80kbps ¹ .
64 kbps Sink / Source	Support for audio stream configurations of 32_1 and 32_2, corresponding to a bit rate of 64kbps ¹ .
Multisink Sink / Source	The ability to send or receive at least two LC3 codec frames per block of an SDU.

Section 11.7 - Interoperability between HAP, TMAP and GMAP

Feature	Description
Multiplex Sink / Source	The ability to send or receive at least two channels of audio, each in separate CISes.
<p>1. These reference settings come from the Audio Stream configuration tables in BAP. Although GMAP defines new audio stream configurations, the timing for the decode will be the same for BAP and GMAS compliant audio streams, as the Presentation Delay is defined after the transport delay.</p>	

Table 11.6 New features introduced in GMAP.

GMAS, which is included in Section 4 of GMAP, allows a client to read the supported GMAP roles and features of a GMAP server. The individual feature support is expressed as a bitmap in a separate characteristic for each of the GMAP roles. Table 11.6 shows the bit map for the UGT role, which is the most complex of the four characteristics describing role features. Supported features are indicated by the value “1”. “0” indicates the feature is not supported. All of the GMAS characteristics are defined in Section 4.7 of the GMAP specification.

Bit location	Description
0	UGT Source feature support
1	UGT 80kbps Source feature support
2	UGT Sink feature support
3	UGT 64kbps Sink feature support
4	UGT Multiplex feature support
5	UGT Multisink feature support
6	UGT Multisource feature support
7	RFU

Table 11.7 The UGT features characteristic

11.7 Interoperability between HAP, TMAP and GMAP

The principal differences between HAP, TMAP and GMAP are in the way that they elevate QoS requirements.

- HAP uses the mandatory requirements from BAP and mandates support for shorter Presentation Delay values for both broadcast and unicast applications.
- TMAP mandates higher quality 32kHz and 48kHz QoS requirements from BAP and mandates support for the same shorter Presentation Delay values as HAP, but only for broadcast.
- GMAP introduces and mandates new QoS settings as well as extending the range of supported Presentation Delay beyond those in HAP and TMAP.

This elevation of requirements can lead to interoperability issues where a mix of HAP, TMAP and GMAP devices are used together. These can be mitigated by considering the potential incompatibilities during the design process and selecting compatible QoS options.

11.7.1 Unicast issues

Bluetooth LE Audio has been designed to offer a guaranteed minimum level of interoperability using the 16_2 and 24_2 codec configurations which are mandatory for all Initiators and Acceptors. If an application supports TMAP or GMAP, then an Initiator will normally prefer to use a higher quality QoS configuration. However, it is not mandatory that it chooses the higher level – it is still permitted to use other configurations, although this may impact the audio experience the user is expecting.

For unicast applications, compatibility should be easy to ensure, as the enumeration process of configuring an ASE will inform the Initiator of the capabilities of each Acceptor. Here, the important features are the Codec_Specific_Capabilities (which can also be checked by reading the PAC records) and the supported range of Presentation Delay values, which are determined from the Presentation_Delay_Min and Presentation_Delay_Max values which are returned after the ASE has completed the Codec Configured operation.

Initiators should not attempt to use a Codec sampling rate or frame size that is not supported. It is entirely up to the application to decide what to use, which may be one of the higher rates mandated by TMAP or GMAP. However, neither mandate that these must be used. An application may choose to configure its audio stream with a lower configuration based on power requirements, other resource constraints or a mismatch in profile support between Acceptor and Initiator. TMAP and GMAP mandate support for these other configurations, but do not mandate their use.

An Initiator should not attempt to configure an ASE with a Presentation Delay outside the range of supported values defined by the Presentation_Delay_Min and Presentation_Delay_Max. It must always use the same value for ASEs of members of a Coordinated Set with audio streams in the same direction. If members of the Coordinated Set have different values for the Presentation_Delay_Min and Presentation_Delay_Max, the configuration value must be the same for all set members with the same audio stream direction. Generally, this will be the highest Preferred_Presentation_Delay_Min value from the ASEs in the Coordinated Set. Note that GMAP specifically chooses different Presentation Delay values for Audio Sinks and Sources, so the choice of value will normally be different for each direction. It should always be possible to configure the ASEs of Acceptors based on the results of the ASE configuration process, providing bidirectional audio streams at a minimum of the 24_2_1 or 24_2_2 configurations.

For TMAP and GMAP, an Initiator that normally wants to multiplex its audio streams may need to make the decision to send the audio streams as individual streams to obtain interoperability, as Acceptor support for multiplexed streams is optional.

None of this configuration process is expected to be visible to the user. However, an

Section 11.8 - Public Broadcast Profile

application could alert them to the use of a “compatibility” mode if the resultant change in audio quality is considered to be noticeable. That is an implementation decision.

11.7.2 Broadcast issues

Ensuring interoperability with broadcast is more difficult, as there is no configuration process. A Broadcast source is set to transmit, with no knowledge of whether its broadcasts will ever be received. If it transmits non-multiplexed audio streams at the mandatory 16 and 24kHz configurations of BAP (and HAP) then it knows that every Acceptor will be able to receive and decode its audio. If it chooses any other configuration, then some Acceptors will not be able to render its broadcasts. For this reason, designers need to think carefully about the requirements of broadcast applications.

Auracast, which is covered in more detail in Chapter 12, mandates that every Broadcast Transmitter which uses the Auracast™ brand must either transmit at the BAP mandated QoS settings, or provide a means for a user to change the broadcast configuration to these from a higher setting, or broadcast at one of the BAP settings in addition to the other configuration, typically by transmitting an additional BIG. The best option will depend on the available resources and airtime. If a Broadcast Assistant is being used, it should filter the list of available broadcasts based on the information it has read from its Acceptor’s PAC Records, and not display or attempt to write the Add Source command for any Broadcast Sources which the Acceptor is incapable of decoding. A Broadcast Assistant may decide to inform the user of incompatible broadcast streams, so that they can ask for a Broadcast Transmitter to be reconfigured.

If a Broadcast Source sets a Presentation Delay value outside the range supported by an Acceptor, the Acceptor should use the closest available value it has, which will be its `Presentation_Delay_Min` or `Presentation_Delay_Max`. An Acceptor should never reject a request to synchronize to a Broadcast because of the value of Presentation Delay set in the BASE of a transmission. It is worth reiterating that the point of the Presentation Delay is to set synchronization in a Coordinated Set. It is not intended to be used to accommodate application specific delays, such as lip-synch. That should be handled by the higher layer applications in the Initiator.

11.8 Public Broadcast Profile

The Public Broadcast Profile (PBP) is a simple, but very interesting profile. It is intended to support the Auracast™ Audio Sharing use case, providing a guarantee that a broadcast Audio Stream is configured such that any Acceptor can receive and decode it. It contains a new UUID – the Public Broadcast Service UUID, which is added alongside the Basic Audio Announcement as an additional Service Type. This means that it appears in an Extended Advertisement, allowing a device to read it without having to synchronize to the Periodic Advertising train and then receive and parse the BASE. This reduces the amount of work for the scanner, helping to reduce its power consumption. Essentially it acts as a filter which a Broadcast Sink or Broadcast Assistant can use to limit its scanning, reducing the power and

time required to identify Broadcast Sources which they know their Broadcast Sink(s) can decode. If a Broadcast Assistant detects a Public Broadcast Announcement that shows only a 48kHz stream, it may display a message on its user interface to inform the user that they should enable its 24kHz sampling configuration to make it accessible.

All Auracast™ devices need to support PBP. We'll look at why and what that means in more detail in Chapter 13.

PBP defines three roles – the Public Broadcast Source (PBS), Public Broadcast Sink (PBK) and Public Broadcast Assistant (PBA). The only requirement on the PBK and PBA roles is that they are able to recognise and interpret the PBP UUID in an Extended Advertisement.

A Public Broadcast Source includes the PBP UUID in its Basic Audio Announcement if at least one of the BISEs in the BIG it is currently configured to transmit is encoded using one of mandatory QoS settings defined in BAP for a Broadcast Sink, i.e., 16_2_1, 16_2_2, 24_2_1, or 24_2_2. The BIG may contain BISEs encoded at other QoS settings, but the PBP UUID should only be transmitted when a BIS with these mandatory settings is active.

The Public Broadcast Announcement also indicates whether the broadcast Audio Stream is encrypted. Either none or all of the broadcast streams can be encrypted.

The Public Broadcast Announcement also mandates the inclusion of the Broadcast Audio Name AD Type, which is a human readable string giving a name to the BIG containing the BISEs. The name does not need to be unique, but is the name that is normally displayed in the UI of a Broadcast Assistant to help a user decide which stream they want to select when multiple broadcast streams are available. It is recommended that the Broadcast Name is a descriptive one which helps the user to make a decision about their choice of Broadcast Source.

11.8.1 LTV metadata

The Bluetooth LE Audio specifications include metadata, which can be included in various announcements to provide more information about an audio stream. These are listed in Section 6.12.6 of the Assigned numbers. We've already looked at the Streaming Audio Contexts LTV and the CCID_List LTV. Most of the others, which are shown in Table 11.8, are used in the Public Broadcast Announcement (PBAnn⁷³), the BASE or a Broadcast Receive State Characteristic (BRSC) to provide more information about the contents or state of an Audio Stream.

⁷³ The abbreviations PBAnn and BRSC do not exist in the specifications. I've introduced them solely for us in Table 11.8.

Section 11.8 - Public Broadcast Profile

AN ref	Name	Description	Used in
6.12.6.3	Program_Info	Details of a stream, typically the program source, or name of a program.	BASE PBAnn
6.12.6.4	Language	Language of an audio stream.	BASE
6.12.6.6	Parental_Rating	Recommended minimum age for listeners.	BASE
6.12.6.7	Program_Info_URI	A URI where further information on a program can be obtained.	BASE
6.12.6.10	Audio_Active_State	Information on whether the audio stream is being transmitted. This is used where audio may be turned off, but the extended advertising train remains active.	PBAnn BASE
6.12.6.11	Broadcast_Audio_Immediate_Rendering_Flag	An instruction to receivers that they should render the stream as early as possible, regardless of the value of the Presentation Delay. Generally, this is used where there is a live stream and the user may be able to hear the ambient sound at the same time.	PBA
6.12.6.12	Assisted_Listening_Stream	Denotes a stream which has been preprocessed to assist listening intelligibility.	PBAnn BASE
6.12.6.13	Broadcast_Name	The same content as the Broadcast_Name AD. This metadata is used to pass the Broadcast_Name data between devices, typically by notifying it from a Broadcast Receive State characteristic.	BRSC
<p><i>Note: The order of the "Used in" column lists the order of elements which are most likely to use it.</i></p>			

Table 11.8 The most commonly used Metadata LTV Types

11.9 The Broadcast Audio URI (BAU)

The Broadcast Audio URI (BAU) is the most recently published Bluetooth LE Audio specification. It is an unusual specification, as it doesn't describe anything that is transmitted over a Bluetooth link. Instead, it defines a data format for out of band transfer of BASS information. It can be used by a Broadcast Assistant or a Scan Delegator to direct a Broadcast Sink to a particular broadcast stream. Its primary use is likely to be in generating QR codes, which may be static printed codes which can be displayed near to a Broadcast Transmitter, or dynamic ones, such as on a phone display, allowing users to scan them to automatically connect to and render that stream.

As we saw in the discussion of Broadcast Streams, Broadcast Assistants allow users to scan for available broadcasts and select which one they want to listen to from a list of available broadcasts. That is an acceptable user experience when there are only one or two broadcasts within range, but as the numbers grow, the list becomes extensive, making the selection a more difficult task. It's a similar issue to what has happened with Wi-Fi, where users can be presented with dozens of alternatives. For Bluetooth LE Audio broadcasts, the issue is further compounded by the fact that some Broadcast Sources may transmit in multiple different languages or with different audio quality configurations, widening the choices that users need to make.

The aim of the Broadcast Audio URI is to simplify the task of choosing a broadcast. It captures all of the information that is needed to identify a broadcast into a single data string which can be provided by an out of band method to a Broadcast Assistant, or directly to a headset. This can be decoded and used to direct Acceptors to synchronize with the broadcast which it represents.

The first implementation that is expected to come to market is to use it for QR codes, which can be scanned by a phone. The phone will decode the contents of the URI and send these to the Acceptor's Broadcast Audio Scan Control Point characteristic using an Add Source or Modify Source operation, instructing it to synchronize to that broadcast stream. Figure 11.10 show how it works, where a smartphone operating as a Broadcast Assistant scans a QR code displayed with a TV, then sends the detail of the TV broadcast to a headset.

Section 11.9 - The Broadcast Audio URI (BAU)

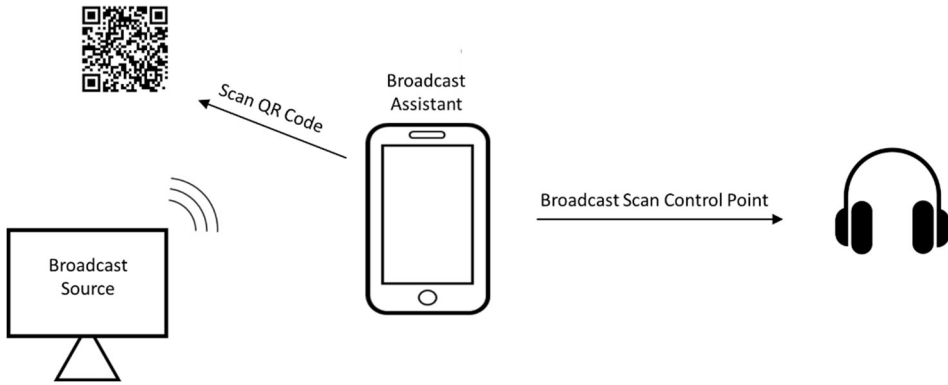


Figure 11.10 An example of scanning a QR code to select a broadcast stream

The QR code could be printed and displayed in the room, or be an overlay on the TV screen. In this example, it could be used with a personal TV, a set of public TVs in a bar, or a TV in a hotel room. If there are multiple TVs, each would have its own QR code. If they have multiple language streams available, each language could also have its own QR code.

Figure 11.11 shows how this could be used in a sports bar, where the QR codes identify the type of sport being displayed on the three TV channels. If the user ran the Broadcast Assistant application on their phone, it would result in all three being displayed (along with any other Broadcast Sources within range), after which the user would need to select their preferred one. Scanning an individual QR code – in this case “Hockey”, would immediately tell the user’s headset to connect to that audio channel.

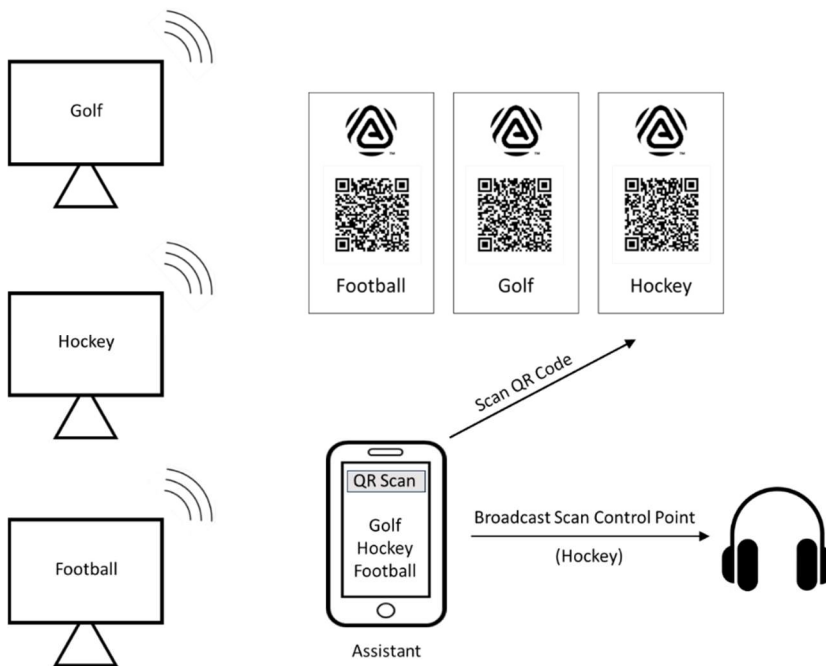


Figure 11.11 Example of QR codes for multiple TVs in a sports bar

This example shows how a venue can use any name to help users make their decision. Here, it is the type of sport being shown, but it could equally be a name given to a TV, where the content changes. It is up to venue owners to choose what is most appropriate to their audience. It is expected that QR code scanning will be included in most Broadcast Assistant apps on smartphones.

11.9.1 The Broadcast URI format

The construction of the Broadcast Audio URI is very straightforward and shown in Table 11.9. It starts with the prefix “BLUETOOTH:” Currently, this is the only defined Bluetooth URI, but the prefix may be used for future URIs as well. To indicate that this is the Broadcast Audio URI, it is followed by the UUID for the Broadcast Audio Scan Service, which is 184F. (Leading zeros are omitted to minimize the length of the URI string).

Element	Item	Value	Description
1	Prefix	BLUETOOTH:	
2	Bluetooth UUID	184F;	Broadcast Audio Scan Service
n	Identifier:Data	See BAU specification	“n” Identifier Data elements followed by “;”
n+3	URI Terminator	;	

Table 11.9 Structure of the Broadcast Audio URI

Following these two items, the URI comprises a list of Identifier | Data pairs, with each pair terminated by a semicolon. At the end of the list of pairs, the URI is terminated with another semicolon.

The Identifiers are two letter codes which cover the bulk of parameters defined in the Broadcast Audio Scan Service for use with the Broadcast Audio Scan Control Point characteristic, as well as a number of metadata LTV elements which can be used with it. The BAU specification defines their format, which is designed to minimise the length of each data element. Only one of the Identifier|Data pairs is mandatory to include, which is the Broadcast_Name. This is used as the primary identifier of the Broadcast Transmitter. If the Broadcast Address is static, the Broadcast ID should also be included, and if the Advertiser Address is static, that should also be present, along with the Advertiser_Address_Type, which will be set to indicate it is static. These are the primary elements which a Broadcast Assistant or Acceptor can use to determine which Broadcast Source to connect to after its subsequent scans.

All other Identifier|Data pairs are optional, but can be used to provide further detail to help select which BIG, or BIS(es) within a BIG a Broadcast Receiver should synchronize to. These allow URI strings and hence QR codes to be designed to select different BIGs or subgroups within a BIG, such as generating separate QR codes for different languages for attendees at a conference or cinema, as illustrated in Figure 11.12.



Figure 11.12 Examples of QR codes for different language streams from the same Broadcast Transmitter

The Broadcast Audio URI defines an Identifier|Data pair for the value of the Broadcast_Code for a BIG. This removes the need for a user to enter a Broadcast_Code using a secondary out of band method. However, it makes it publicly available for anyone who has access to the QR code. If this is not an issue, it removes a further step from the user selection process.

QR codes are particularly useful for session based applications, such as sharing audio from a smartphone, or inviting friends to share TV audio. An example is shown in Figure 11.13, where an audio sharing app on a smartphone generates a dynamic QR code to allow the phone user to share their current audio stream with friends.



Figure 11.13 An example of a QR code generated to allow sharing of audio content from a smartphone.

An important point to remember is that the Bluetooth URI does not include any timing information. This means that either the Broadcast Assistant that receives it, or the Acceptor(s) must perform a scan to determine where to find the BIG. If the Broadcast Assistant supports scanning for extended advertisements, it can use PAST to transfer everything that the Acceptor needs in order to synchronize. If it does not, the Acceptor will need to scan to obtain the BIGInfo, which contains that information.

-oOo-

That concludes our coverage of the Bluetooth LE Audio specifications. In the remaining chapters we will look at the way they can be used to change the consumer experience of audio with completely new applications. The most important of these is a new broadcast ecosystem which the Bluetooth SIG has branded as Auracast™.

Chapter 12. Auracast™

At the beginning of the Bluetooth® LE Audio specification development, a key requirement was to support broadcast audio. That came from the hearing aid community, who were looking for a way to complement telecoil inductive loops, but offering easier and more scalable installation, as well the ability to support multiple, overlapping audio streams. As the specification work progressed, more and more consumer audio companies began to see the potential in broadcast and joined the development process. Today, the broadcast feature in Bluetooth LE Audio is seen by many as its most important feature, bringing shared audio to everyone – not just hearing aid users. As developers start to understand its benefits and roll out new audio experiences that use its features, it is fair to say that it could be the biggest step change in consumer audio since the introduction of stereo in the late 1950s.

As with many such new technologies, it requires designers to rethink the way they make products. Most of the history of audio has been one of incremental changes – regular, small enhancements to audio quality as a result of better components and more efficient codecs. Digital technology has helped change the way that audio content is delivered, initially in the move from cassettes and vinyl to CDs, but also enabling access to streaming media. Alongside these changes, the user experience has become increasingly personal, with an audio source streaming to a single set of speakers, headphones or earbuds. The recording industry has a vested interest in preserving that model, as it is one where every user purchases individual access to the same content. That business model has generated a developer mindset which has been slow to appreciate the new freedoms which broadcast audio brings and the design choices which need to be made to make the most of them.

To drive the new usage model, users need to be able to select from multiple different broadcast streams. The classic Bluetooth paradigm was to centre everything around a mobile phone, which became a user's sole source of audio. Users could access a picklist if they wanted to change from earbuds to speakers, but the experience is predominantly one of changing the rendering device, and is almost always initiated by the phone. In contrast, broadcast audio sees the phone as just one of many potential audio sources, with decisions made by the earbuds, or a device acting as their proxy, effectively extending their user interface. That's the role of the Broadcast Assistant, which we introduced in Section 8.6.1. It may be implemented as an app on the same mobile phone, but it could take many alternative physical forms. This concept of migrating the control of what you are listening to over to the device which is rendering the audio, is referred to as the “Sink led journey”. It lies behind the introduction of announcements and the unconnected model, which support this new versatility.

With the prospect of multiple public broadcast sources, alongside personal ones, there is a need to ensure a minimum common set of features which can guarantee interoperability for all users. These need to encompass devices like hearing aids, where extended battery life is paramount, as well as personal devices, where the industry prioritises the highest possible Bluetooth LE Audio quality. To address these, the Bluetooth SIG has developed the Auracast™ brand, which lays down guidelines for broadcast audio sources to try and ensure

interoperability. The Auracast™ name and logo can be used to market products which meet the Auracast™ Simple Transmitter Best Practices Guide. It also acts as a public statement of the availability of a compliant audio broadcast, in a similar way that the telecoil and Wi-Fi logos have been used to indicate the availability of a wireless service. The Auracast™ logo is shown in Figure 12.1. Details on the requirements to use it are explained in the Brand Guide for Bluetooth Trademarks⁷⁴.



Figure 12.1 The Auracast™ Figure Mark

12.1 The basic principles

To help understand the reasons for these guidelines, it's useful to look at a few of the envisaged use cases for Auracast™. These illustrate some of the compatibility issues which need to be addressed to ensure a common user experience, helping consumers have confidence that an Auracast™ branded product will always work.

12.1.1 The personal TV

One of the simplest use cases is where two people are listening to a TV, as shown in Figure 12.2. One of them is using earbuds, whilst the other is wearing hearing aids.

There are a number of different ways this can be implemented. The two simplest are:

- A Broadcast Transmitter integrated in the TV, so that a user can select Auracast™ transmission from the TV's menu or a remote control.
- A stand-alone Auracast™ Transmitter which is plugged into one of the audio outputs on the rear of the TV.

On the surface it looks very simple, but there are a couple of important subtleties. The first is that the two users may want to receive different streams. The second is that they may also be within range of other Broadcast Transmitters, so need a way to make sure they are receiving the correct broadcast stream.

⁷⁴ Branding guides are available at <https://www.bluetooth.com/marketing-branding/>

Section 12.1 - The basic principles

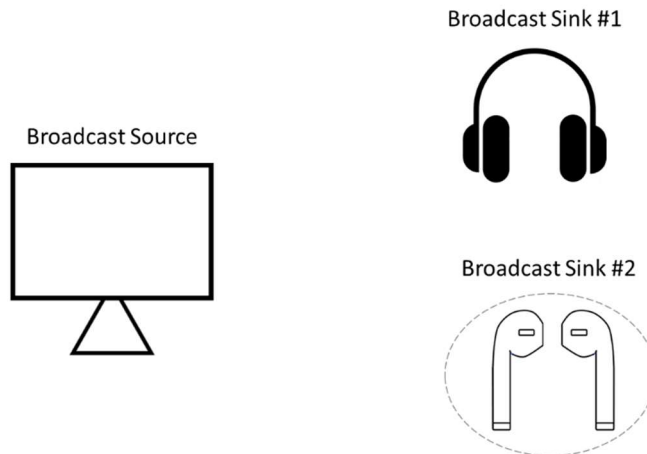


Figure 12.2 A simple TV broadcast audio use case

Starting with the requirement for multiple broadcast streams, there are several different reasons why the two users may want to receive different audio streams. The first is for intelligibility. If you're wearing a hearing aid, your primary concern is usually to hear the dialogue from the TV program as clearly as possible. To assist in this, an increasing number of TV programs include "hearing enhanced" or "dialogue assist" streams, where the audio has been mixed to lessen background noise and enhance speech. This makes it easier for a listener with hearing loss to understand the conversation.

The second reason is that an increasing number of programs are available with multiple language tracks. In both cases Bluetooth LE Audio allows multiple Bluetooth LE Audio streams to be broadcast simultaneously, so that different users can select what they want to hear⁷⁵.

A more subtle, third consideration driving the use of multiple streams is that most hearing aids don't have the resources to decode the High Quality codec configurations that TV vendors like to set as default. To decode these QoS packets would have a detrimental effect on the hearing aids' battery life. Therefore, all Auracast™ compliant TVs need to include an accessible listening mode to balance the needs of all users and not exclude those with hearing loss. This may mean that a TV needs to be capable of broadcasting the same content at two different QoS configurations.

12.1.2 Public venues

Auracast™ is set to revolutionise the availability of assisted hearing in public venues. Today, the physical difficulty of installing telecoil loops means that often they are only fitted during

⁷⁵ This does require TV and video streaming adaptor designers to realise that they should allow multiple Bluetooth LE Audio outputs to be present at the same time. At the moment, most TVs disable ambient audio when Bluetooth is selected and only provide one audio stream. This is explained in more detail in Chapter 13.

the initial building phase or major refurbishments. Auracast™ can offer the same facility with a simple, portable device approach, which can be installed at any time.

Figure 12.3 shows a handheld microphone that includes a Broadcast Transmitter. It provides a mono audio stream which is received and rendered by a number of users. They can be wearing a single, or pair of hearing aids, headsets or earbuds. The same broadcast audio signal can also be used to drive the PA speakers in the venue, providing a complete, low latency audio system for everybody in the room. The microphone of Figure 12.3 can be easily replaced by a stand-alone Auracast™ Transmitter which plugs into the audio output of an existing sound desk or audio system.

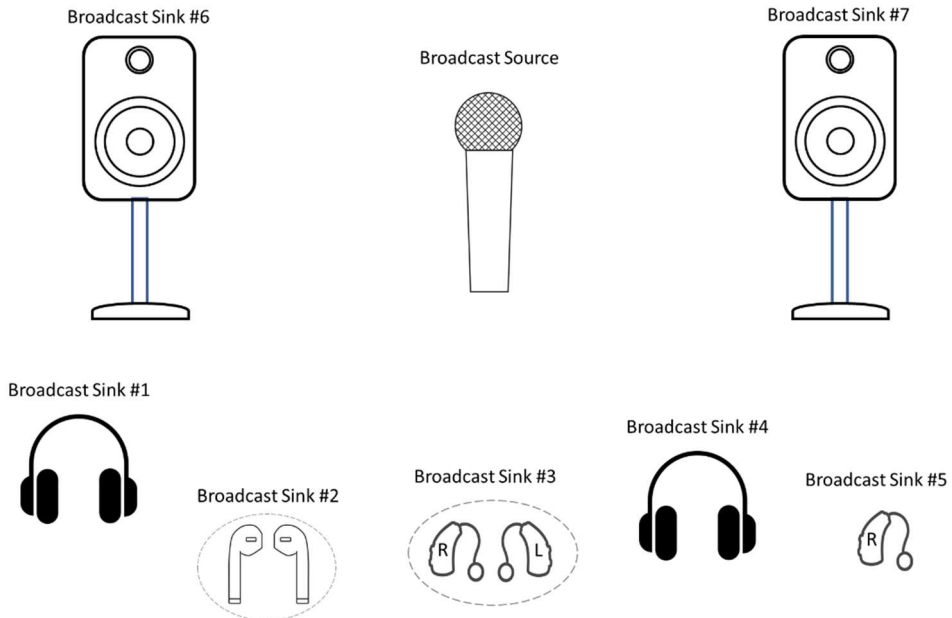


Figure 12.3 A typical public Auracast™ deployment

It illustrates the simplicity that Auracast™ brings to public audio. There are no cables, no mixing desk and no induction loop. The entire PA system and Auracast™ coverage can be installed in just a few minutes in any venue, and will be up and running as soon as it is turned on. It will transform audio installations, as it is a highly affordable way of making venues accessible for a full range of user audio requirements.

As a third example, we have the scenario usually described as the “gym” or “sports bar”, where there are multiple silent TVs showing offering different programs. Users with Auracast™ earbuds or hearing aids can choose which of the multiple channels they want to listen to.

Section 12.1 - The basic principles

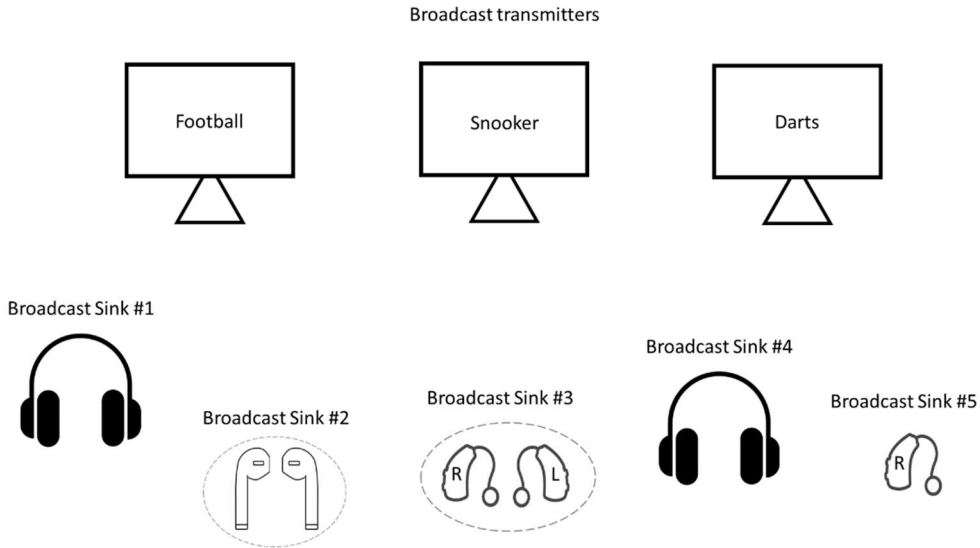


Figure 12.4 A sports bar or gym with multiple silent TVs

Figure 12.4 illustrates the use case with three TVs, each showing a different channel. Again, any number of users can select the channel they want to listen to. As all of these transmitters are serving the same area, users need a means to select the correct broadcast channel. The simplest way to do that would be to use the Broadcast Audio URI that we covered in Section 11.9 to provide QR codes for each of the different programmes, as shown in Figure 12.5. These could be displayed in any convenient location.

To start receiving their preferred stream, a user just needs to scan the corresponding QR code with their phone. Alternatively, they could perform a scan for the broadcast information with their Broadcast Assistant, but that requires more user interactions.

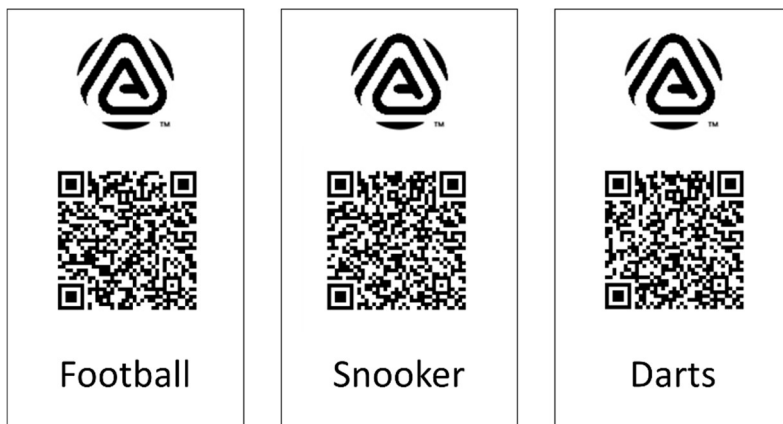


Figure 12.5 An example of QR codes for multiple screens in a sports bar

There may be additional complexity in a scenario like this. Each TV could present multiple different audio channels to provide alternative QoS settings or different languages. There are

also practical considerations. Although domestic TVs, such as the example in Figure 12.2, are likely to contain an internal Auracast™ Transmitter, in a venue such as the gym or sports bar, there are likely to be multiple screens showing the same video content. Rather than having a separate transmitter in each TV, the transmissions are more likely to come from a central control box which includes multiple Auracast™ Transmitters and also distributes the video streams to multiple displays. That is up to the venue owner. The same companies that provide multiple screens for public use today are likely to add managed Auracast™ infrastructure as part of their future, commercial offering.

There are many more applications for Auracast™ in TV applications, and we'll look at some of them in the following chapters, but the ones we've just covered encompass the main design issues that implementers need to address to achieve global interoperability, giving customers confidence in choosing and using Auracast™ wherever they see the name or logo displayed on a product or at a venue. These start with requirements for Broadcast Transmitters, which have been published in the Auracast™ Simple Transmitter Best Practices Guide.

12.2 The Auracast™ Simple Transmitter Best Practices Guide

To ensure the promise of interoperability is met, the Bluetooth SIG has produced the Auracast™ Simple Transmitter Best Practices Guide, which lists the features a Broadcast Transmitter must support in order to use the word Auracast™ to describe or promote itself. The guidelines do not define new features, but mandate a combination of features and behaviours across the Bluetooth LE Audio specifications that an Auracast™ branded product must support. To quote from its introduction, it “imposes certain design principles on broadcast audio devices, including selected audio stream parameters which are common to all products. These design principles help ensure global interoperability between all kinds of rendering devices and both public and personal Auracast™ Transmitters”.

Going back to the basic premise of Bluetooth LE Audio broadcast, a transmitter has no knowledge of whether any Acceptor is listening to its audio streams, or the level of capabilities of any Acceptor which is attempting to receive them. All it can assume is that they support the minimum mandated QoS requirements, which means that they can receive, decode and render non-multiplexed LC3 streams with sampling rates of 16kHz or 24kHz and a 10ms SDU. A Broadcast Transmitter that uses these settings will be globally interoperable. Auracast™ streams may be encrypted, depending on the application. How the Broadcast_Code is provided for an encrypted stream is not specified.

12.2.1 Standard Quality Public Broadcast Audio

These settings, which BAP mandates for all Acceptors are defined in the Auracast™ Simple Transmitter Best Practices Guide as the Standard Quality Public Broadcast Audio configuration, using the “Standard Quality (SQ)” definition from the Public Broadcast Profile (PBP). Every Broadcast Transmitter that claims Auracast™ compatibility must support transmissions at this configuration. If they do not use them as their default, they must support them on demand.

Section 12.2 - The Auracast™ Simple Transmitter Best Practices Guide

12.2.2 Public Auracast™ Transmitters

The Best Practices Guide goes on to define a Public Auracast™ Transmitter, which is intended for deployment within a public venue, such as commercial public address (PA) systems, televisions and audio streamers. The default configuration for these devices must include a Standard Quality Broadcast Audio stream. This should ensure that wherever an Auracast™ logo is displayed in a public space, every hearing aid, headset and headphone user can be assured of the presence of a stream which their device can receive. The intention is that in public spaces, the Auracast™ logo has the same universal interoperability meaning as today's telecoil logo.

12.2.3 High Quality Public Broadcast Audio

Although 24kHz sampling with LC3 results in an audio quality that listeners generally consider to be more than acceptable, many product vendors want to use higher sampling rates to differentiate their products. That resulted in the inclusion of the "High Quality (HQ) configurations in the Public Broadcast Profile, which are also incorporated in the Auracast™ Simple Transmitter Best Practices Guide. However, they come with some limitations on their use.

12.2.4 Personal Auracast™ Transmitters

To allow the Auracast™ branding to be applied to products that support the High Quality Broadcast Audio configuration, the Best Practices Guide defines a Personal Auracast™ Transmitter, which supports the High Quality (HQ) configurations. These are products which are intended for personal use, such as smartphones, tablets, laptops, PCs, home TVs, or home audio streamers. The definition comes with a caveat. Personal Auracast™ Transmitters may default to transmitting a High Quality (HQ) Public Broadcast Audio stream but must allow the user to select a Standard Quality (SQ) broadcast audio stream so that they can be accessed by devices which are not capable of decoding and rendering a 48kHz sampled stream. The intention is that there should be an easily accessible method to allow the product's owner to configure a Personal Auracast™ Transmitter to broadcast the same audio content at one of the Public Standard Quality configurations. This may be accomplished by changing its transmissions to the lower sampling rates required for SQ, or transmitting both HQ and SQ streams concurrently. Depending on the QoS settings and the available resources on the transmitter, this may require the current BIG to be terminated and a new BIG established to replace it. Transmitting a mono, SQ stream by itself, or alongside the HQ stream(s) is sufficient to meet the Auracast™ requirement.

How this is done is up to the implementation. However, unless a Broadcast Transmitter can do this with a simple user interaction, an HQ Broadcast Transmitter is not allowed to brand itself as Auracast™.

12.2.5 Advertising information

A second strand to the Auracast™ Simple Transmitter Best Practices Guide is to ensure that a transmitter provides sufficient information to allow a scanning device to recognize it as an Auracast™ compliant transmitter. This requires certain information to be present in a number of AD Types and Announcements in the Extended and Periodic Advertisements. Figure 12.6 provides an overview of these elements and where they are located.

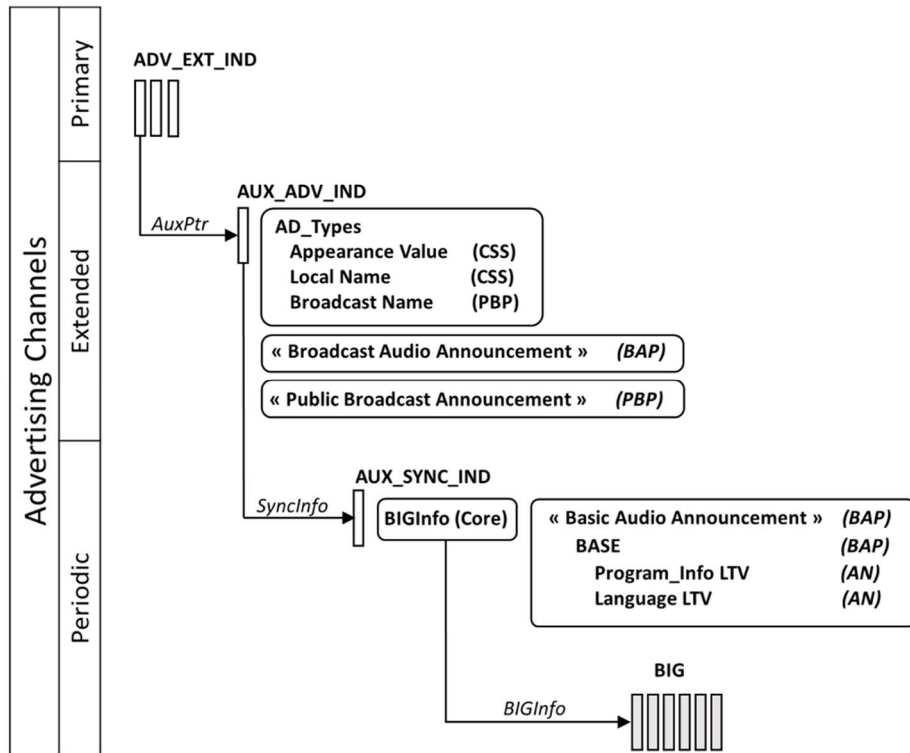


Figure 12.6 The location of Auracast™ related data in Primary, Extended, and Periodic advertisements

All Auracast™ Transmitters must include:

- A correctly populated Public Broadcast Announcement. This allows scanners to detect Auracast™ compliance without having to receive and parse the BASE in the Periodic Advertising train.
- The Broadcast_Name AD Type, which provides the Broadcast Source's name, which a Broadcast Assistant can use in its user interface, and
- A BASE structure in the Basic Audio Announcement which includes a subgroup which contains an SQ or HQ configuration.

The Best Practices Guide also includes recommendations for other advertising content parameters, using Metadata LTV structures.

Section 12.3 - The Broadcast Assistant

12.2.6 Additional transmitter information

The Auracast™ Simple Transmitter Best Practices Guide also covers a number of physical transmitter settings, including advertising interval, output transmit power level and broadcast audio level. These help to provide a uniform user experience in terms of coverage and the time required to discover the presence of an Auracast™ stream. They also remind implementors to avoid using multiplexed audio streams, as it is not mandatory for Broadcast Receivers to support them. Where Auracast™ Transmitters need to support stereo, it should be sent as individual left and right streams.

12.3 The Broadcast Assistant

It has been interesting to see how ingrained the existing peer-to-peer Bluetooth topology has been in developer's thinking. The new broadcast experience means that users will often be within range of multiple Broadcast Transmitters, each of which may be providing multiple different audio streams. It would seem obvious that users need a way to select the one they want to listen to, which requires the existence of a Broadcast Assistant to complement earbuds, hearing aid and headphones. Despite that, companies have been slow to develop Broadcast Assistants, resulting in an ecosystem that has a key part missing.

Part of this is probably due to the same historic view of topology, where phone manufacturers believe that they are in control of a largely passive rendering device, and earbud vendors expect phones and PCs to provide the means to select what they should do. The consequence of this mindset has led to companies producing products that don't expose or utilize the full capabilities of the broadcast ecosystem. That's probably not unexpected for the first tranche of a new experience. At some point, the key phone and PC operating systems will integrate full Broadcast Assistant functionality. In the meantime, there are ways that manufacturers can innovate to remove this barrier.

12.3.1 Broadcast Assistant Form Factors

Most developers working with Bluetooth LE Audio have made the assumption that Broadcast Assistants will be implemented as apps on smartphones. In the course of time, that will probably be true, with every smartphone incorporating a native Broadcast Assistant, such as the example in Figure 12.7.

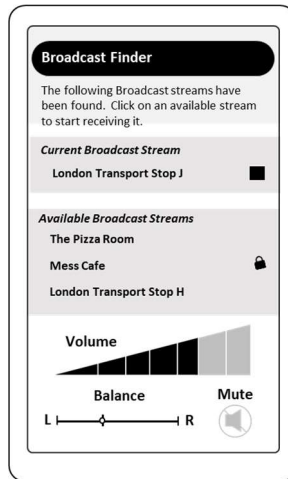


Figure 12.7 Example of a Broadcast Assistant user interface on a smartphone

A phone app is an obvious way to implement these features, but it is a rather limited view of the possibilities for a Broadcast Assistant. Any device which has a Bluetooth chip that supports scanning for extended advertisements and pairing to a Coordinated Set can be a Broadcast Assistant. An interesting approach to a Broadcast Assistant is building one into a battery case, like the mock-up of Figure 12.8.



Figure 12.8 An example of a battery case incorporating a Broadcast Assistant

Here, the battery box includes a small touch display for controlling its earbuds. It can be used for volume control, as well as scanning to find local Auracast™ Transmitters, displaying their details, and letting the user select the one they want to listen to. A particularly nice feature of this approach is that the Broadcast Assistant can be paired with the earbuds during manufacture. This means that as soon as the user takes them out of the battery box, they can scan for Auracast™ Transmitters and connect to an audio stream with no other setup procedure. It's about as good a user experience as it's possible to get.



Figure 12.9 Examples of different form factors for Broadcast Assistants

Figure 12.9 shows a few more examples of possible implementations, showing a watch as a Broadcast Assistant, a simple remote control which lets users cycle through available Auracast™ Transmitters and a basic e-ink display which could be worn as a lanyard. Simple devices like these can be implemented with a single chip, giving implementers an amazing degree of freedom for new means of audio control.

A user can have multiple Broadcast Assistants, with any number of them active at the same time. As they are designed to be interoperable, users can mix and match Broadcast Assistants from any manufacturer. All Broadcast Assistants must provide the same basic search and selection functionality, allowing users to find and receive Broadcast audio streams.

12.4 Auracast™ Assistant and Receiver Guides

The Auracast™ Simple Transmitter Best Practices Guide is a fairly comprehensive design document, so it surprises many designers to discover that there are currently no similar guides for Auracast™ Receivers or Auracast™ Assistants. The reason for that is that Auracast™ is built around the mandatory requirements that all Broadcast Receivers and Assistants must implement from the BAP, BASS, PBP and CAP specifications. That is why Broadcast Transmitters should not deviate from the requirements of the Auracast™ Simple Transmitter Best Practices Guide, as other features may not be universally supported by Broadcast Receivers.

However, even with the mandatory features, there are some basic design principles that should be followed to help with the Auracast™ experience, particularly around scanning and the way that information about broadcast audio streams and transmitters are stored, as well as the use of Audio Locations.

12.4.1 Maintaining a database of available Broadcast Transmitters

BAP mandates that every Broadcast Receiver must be able to scan for available broadcast streams. If we consider a simple user interface, that could be initiated by a user pressing a

button on a single hearing aid, in the same way that they currently do for telecoil⁷⁶. As the order in which Broadcast Transmitters are discovered is random, it's probably not a good design choice to synchronize with the first one which is found, rather than completing a scan and then making a connection choice. That connection choice may be to select one that has previously been used, or the one with the strongest signal – that decision is down to implementation. However, if your device is going to make a choice it needs to build up a database of what it has discovered in order to provide the required user experience.

The most obvious way of storing this information is for a Broadcast Sink to populate a set of Broadcast Receive State characteristics for each of the Broadcast Transmitters that it finds during its scan. In the early days of Auracast™ an earbud may only find a single transmitter, but the number will grow. Hence it's good practice for a Broadcast Sink to support at least six or more instances of the Broadcast Receive State characteristic.

At this stage, the Broadcast Receiver does not need to synchronise to each PA. It knows each transmitter's Source Address, Advertising SID and Broadcast ID, which may be enough to make a decisions if the connection strategy is to reconnect to known sources. It should also read the Public Broadcast Announcement, which will tell it the Broadcast Transmitter's name and whether it is Auracast™ compliant. The Broadcast Name should be stored in the Metadata for each subgroup in the Broadcast Receive State characteristic.

As each new Broadcast Receive State characteristic instance is populated, it will be notified. If the Broadcast Receiver is a member of a Coordinated Set, and a Broadcast Assistant is present, these values should be written to the Broadcast Receive State characteristics of the other set member. If that is also scanning, the Broadcast Assistant should ensure that Broadcast Receive State characteristics in both Broadcast Receivers contain the same information⁷⁷.

Each active Broadcast Assistant may also add additional data that it has acquired if it is scanning on behalf of either of the Broadcast Receivers. This emphasises the value of both Broadcast Assistants and Receivers implementing multiple Broadcast Receive State characteristic instances, as it allows all three devices to have a consistent picture of available Broadcast Transmitters.

As users move around, they will come in and out of range of Broadcast Transmitters. That means that Broadcast Receivers need a strategy to manage their Broadcast Receive State lists to make sure they remain fresh. This is implementation specific and can be done by a Broadcast Receiver, or be delegated to a Broadcast Assistant. As this is implementation

⁷⁶ There's no reason why the existing telecoil button can't have a dual function – selecting an available telecoil loop if one is discovered and then scanning for Auracast™ Transmitters if no telecoil loop is found.

⁷⁷ Their Source_ID values may be different as each is assigned locally by the Broadcast Sink. A Broadcast Assistant may also assign different BIS_Sync values, based in the Audio Sink Locations value of each Broadcast Sink.

specific, there is an argument for these checks to be performed by the Broadcast Receiver, as it has no way of knowing whether a Broadcast Assistant is performing this task on its behalf. Only one member of a Coordinated Set needs to perform this check, as its notifications should be relayed to the other set members by its Broadcast Assistant.

12.4.2 Identifying and selecting broadcast channels

With broadcast, once a Broadcast Transmitter is selected, the Broadcast Receivers need to decide which BIS each of them needs to connect to. Assuming that only one BIG is available, there may be more than one subgroup, with each subgroup containing more than one BIS. This is where the Metadata LTVs are used to provide information for Broadcast Assistants and Receivers to make a choice of BIS.

12.4.3 Channel Allocation in Broadcast Transmitters

A Broadcast Transmitter uses metadata to help a Broadcast Receiver decide which BIS to choose. It does this using metadata in Level 3 of its BASE structure to identify which Audio Location each BIS is intended for, by including the Audio_Channel_Allocation LTV.

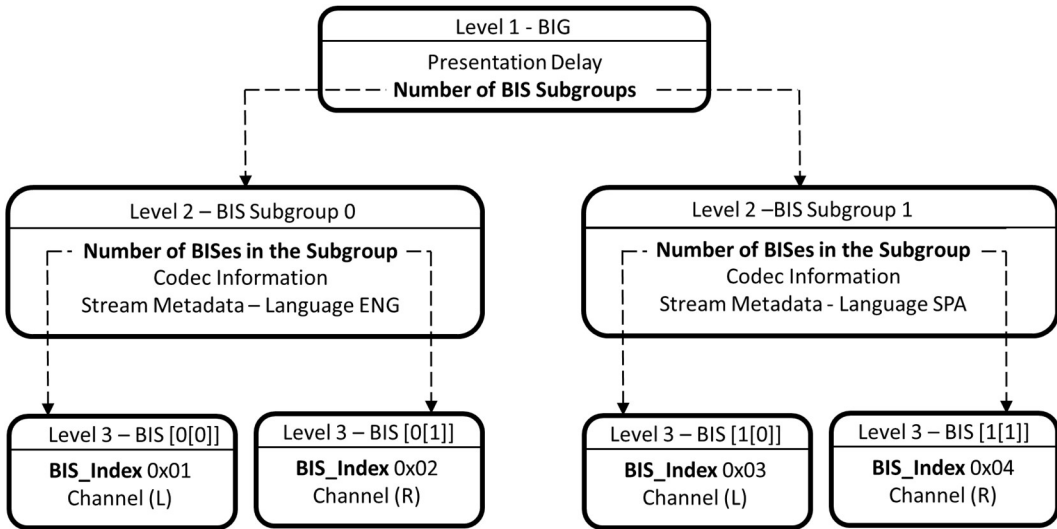


Figure 12.10 BASE structure for two subgroups with stereo streams in different languages

The Metadata in Level 2 of the BASE provides further data for BIS selection. Figure 12.10 shows a BIG with both an English and Spanish stereo stream. As a subgroup should only contain BISes which are expected to be rendered as a set, the English and Spanish streams are separated into two subgroups.

Figure 12.11 shows the case where there is only a single language, but a mono stream is available which has been enhanced for increased intelligibility as well as the stereo content. As this would not normally be rendered in the same device (or pair of devices), it is in its own subgroup. The metadata at Level 2 identifies this as an Assisted Listening Stream (ALS). A BIS containing a single mono audio stream should not include an Audio_Channel_Allocation LTV. However, if scanning devices see an Audio_Channel_Allocation of 0x00000000, they

should interpret it as mono, as this has been seen in some early implementations⁷⁸.

The Metadata LTVs can include further information to help Broadcast Assistants and Receivers make sensible choices. As the Metadata LTVs are defined in Assigned Numbers, new options may become available between major specification releases. Scanning devices should ignore Metadata in BASE or the Public Broadcast Announcement that they are unaware of.

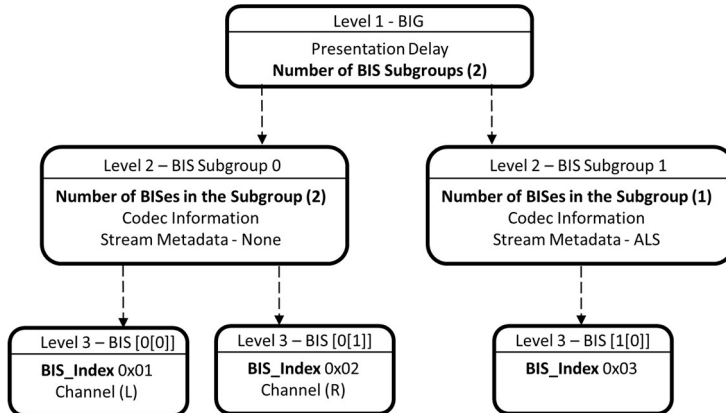


Figure 12.11 A BASE for a stereo stream with an additional Assisted Listening mono stream

12.4.4 Channel Selection in Broadcast Receivers

Each Broadcast Receiver should have an Audio Sink Locations characteristic, which identifies which Audio Locations it would like to receive. If it is designed to support only a single mono stream, this can be omitted, but the absence of a Sink Audio Locations characteristic is equivalent to one with a value of 0x00000000. If a Broadcast Receiver allows the value of its Audio Sink Locations characteristic to be changed, either by letting a Client write it, or through a local action, such as a physical switch which sets a speaker to receiver left, right or mono, then an Audio Sink Locations characteristic must be present, even if its default value represents mono.

A Broadcast Receiver uses the value of its Sink Audio Locations characteristic to select which BIS within a subgroup to synchronise to and render. If there are multiple subgroups in the BIG, the choice of subgroup should occur first. This will normally be based on two items of information:

- The codec configuration, where a device may have a preference for a 16kHz, 24kHz

⁷⁸ The initial version of the Bluetooth LE Audio specifications were unclear about the use of the Audio_Channel_Allocation value for mono streams in Broadcast Transmitters. This has been clarified in the latest release (BAP 1.0.2 and Assigned Numbers), which makes it clear that it should not be present with mono.

Section 12.4 - Auracast™ Assistant and Receiver Guides

or 48kHz stream, and

- The level 2 metadata, which provides information about the content of the stream, such as language and whether it is a hearing enhanced stream.

Remember that we are talking about channel selection here, i.e. which BIS to select from within a subgroup of a selected BIG. In most cases, that results in a left earbud or hearing aid picking the left audio stream and the right earbud picking the right audio stream. Broadcast Receivers should be able to make these decisions autonomously, either because the Sink Audio Locations characteristic values have been set as default by the manufacturer, or the user has been able to configure it in the earbud's setting. When a Broadcast Receiver makes an autonomous decision, it will write that into the BIS_Sync field of the Broadcast Audio Scan Control Point characteristic and the BIS_Sync_State field of the Broadcast Receive State characteristic for the appropriate subgroup of the selected BIG, once it has confirmed successful synchronisation.

Broadcast Receivers can make more subtle decisions, based on dynamic conditions. If a left earbud has a Sink Audio Locations characteristic value signifying Left, it is also capable of receiving a stream labelled as mono. If both audio streams are available, it is up to the implementation to decide which of those two BISes to synchronise to. For example, a hearing aid wearer may prefer that both of their hearing aids receive an Assisted Listening mono stream in preference to separate left and right mono streams. Another example would be the case where the battery in a user's right earbud has gone flat. If the other earbud is aware of this, it might decide to change from its current left audio stream to a mono audio stream to provide a better user experience. Note that these prioritization choices are not exposed to Broadcast Assistants – they are down to autonomous behaviour in the Broadcast Receivers.

12.4.5 Channel Selection from Broadcast Assistants

In most cases, users will make choices from Broadcast Assistants, because it's a richer user interface. These will normally read the PAC Records and Sink Audio Location characteristics from their Broadcast Receivers and use this information to decide what they write to the BIS_Sync field of each device's Broadcast Audio Scan Control Point characteristic, directing the Broadcast Receiver to synchronise to the appropriate BIS. Once the Broadcast Receiver has successfully synchronised, it should update its Broadcast Receive State characteristic and notify successful synchronization.

A Broadcast Assistant does not need to specify a BIS – it can leave this choice up to the Broadcast Receiver. A Broadcast Receiver can also ignore this request and choose a different BIS if its configuration imposes different user priorities, as explained above.

Unlike the unicast case, Broadcast Transmitters and Receivers work independently. A Broadcast Receiver makes its own decisions, helped, but not necessarily based on what a Broadcast Assistant instructs it to do. Designers should consider the case of how a Broadcast Receiver makes these decisions in the absence of a Broadcast Assistant to provide a consistent and logical user experience.

12.5 Virtual Broadcast Assistants⁷⁹

Most developers make the assumption that earbuds and hearing aids with a limited user interface would use a smartphone acting as a Broadcast Assistant to perform scanning and find Broadcast Transmitters. However, that is not the only option. As we saw in Section 12.4.1, not only can a Broadcast Receiver perform its own scanning, it should also maintain a database of what it has found, regardless of who performs the scans. This database of information is held in a set of instances of the Broadcast Receive State characteristic. These are all GATT characteristics, which means that they can be read and written by any Bluetooth LE device, regardless of whether it is capable of scanning. This means that older Bluetooth smartphones, laptops and tablets can control a set of earbuds or hearing aids by reading the Broadcast Receive State instances on their Broadcast Receivers, presenting that to the user to make a selection and then writing to the Broadcast Audio Scan Control Point characteristics in those Broadcast Receivers.

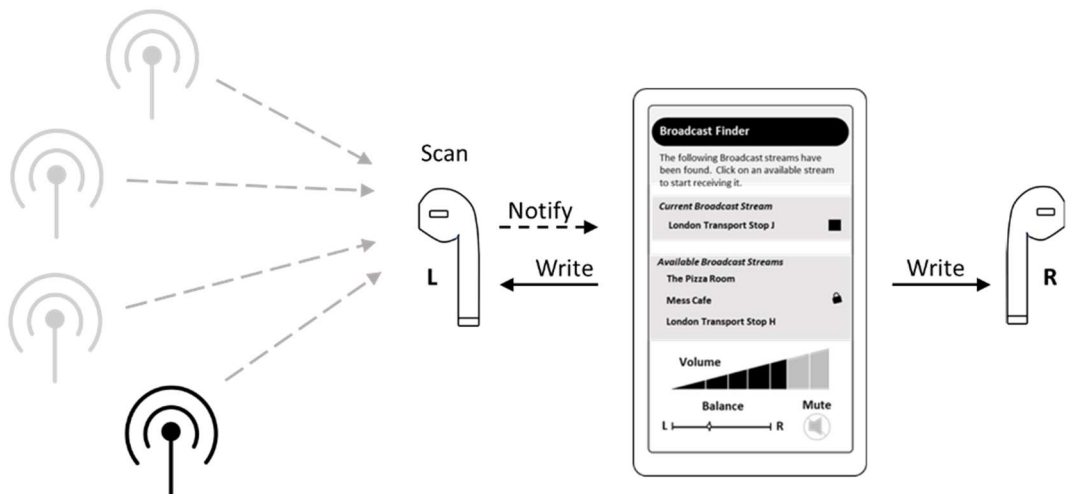


Figure 12.12 An example of the operation of a Virtual Broadcast Assistant

Figure 12.12 illustrates how this works. In this example, the left earbud performs scanning, populating its Broadcast Receive State characteristic instances with the Broadcast Transmitters it has found. The Virtual Broadcast Assistant app on the phone is paired with both earbuds. It does no scanning, but reads these Broadcast Receive State characteristics and displays them for the user. When the user makes a choice by clicking on one of the entries, the virtual Broadcast Assistant app performs a Modify Source operation on the Broadcast Audio Scan Control Point characteristic on both the left and right earbud, directing them to synchronise

⁷⁹ The term “Virtual Broadcast Assistant” is not used in the Bluetooth LE Audio Specifications. It has been coined to describe a device or app which interacts with the Broadcast Audio Scan Control Point characteristic, but does not support scanning for extended advertising.

Section 12.5 - Virtual Broadcast Assistants

to the appropriate BISes within the BIG. (As we've seen above, it could also leave the decision of the choice of BIS to each earbud, by writing 0xFFFFFFFF (no preference) to the BIS-Sync parameter field).

Note that only one of the earbuds needs to perform scanning. An implementation may decide to alternate the scanning between the two earbuds to help equalize battery life between them, but this is an implementation decision. Figure 12.12 also shows that the Virtual Broadcast Assistant can be used to control volume.

Figure 12.13 illustrates the difference between a Broadcast Assistant and a Virtual Broadcast Assistant. The Broadcast Assistant at the top supports scanning, allowing a Broadcast Sink to delegate the scanning role.

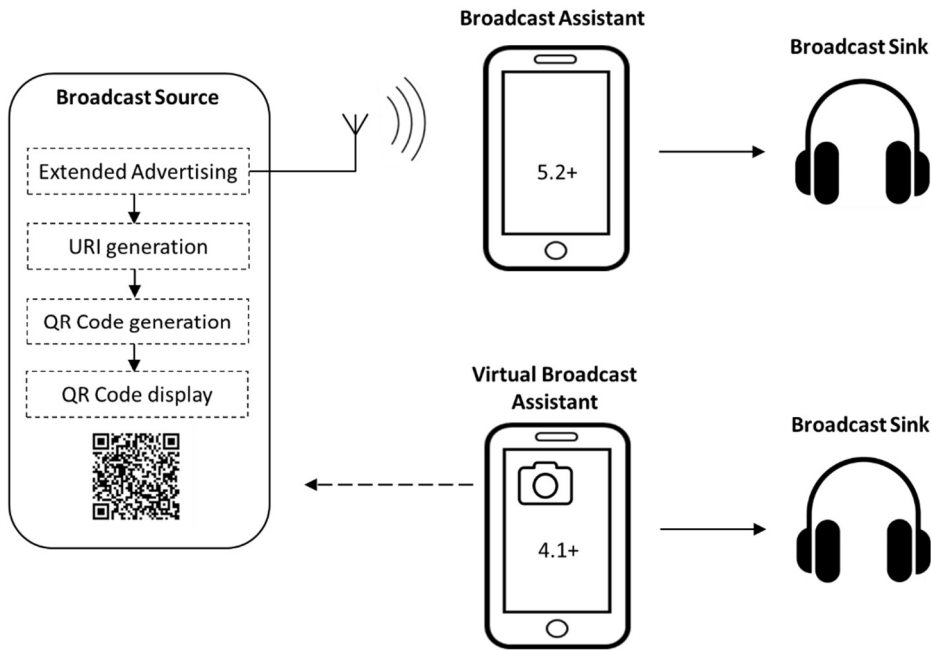


Figure 12.13 A comparison of a Broadcast Assistant and a Virtual Broadcast Assistant

In contrast, the Virtual Broadcast Assistant relies on information from another device to identify the presence of Broadcast Transmitters. This may be by reading the Broadcast Receive State characteristics on its paired Broadcast Sink(s), or in Figure 12.13 by scanning a QR code representing the Broadcast Audio URI on a Broadcast Transmitter. Having written these details to the Broadcast Receiver(s), each Broadcast Receiver will need to perform scanning before they can synchronize to the broadcast Audio Streams.

Chapter 13. Practical considerations

Throughout this book I've stressed that Bluetooth® LE Audio has been designed from the ground up. It's a completely new approach to audio, starting with the Core specification. It has a new codec – the LC3, and a large audio middleware suite - the Generic Audio Framework, which defines how connections are made, maintained and manipulated, as well as including a range of client-server control functions. On top of these, there is a growing set of top-level profiles that define specific configurations of the underlying specifications to support a variety of audio applications.

In an ideal world, developers would have looked at the new Bluetooth LE Audio specifications and written new implementations from scratch. However, that's not the world we live in. Bluetooth Classic is deeply embedded into many platforms and operating systems, such as Windows, MAC OS, iOS and Android, all of which need to continue to support Classic audio applications alongside Bluetooth LE Audio.

The teams supporting these platforms have over twenty years of experience with Bluetooth Classic, which means that some initial implementations are inevitably flavoured with the way that things have been done in the past. In the short term that may mean that features may not work exactly as expected, as developers face the steep learning curve of a completely new way of implementing audio. In time, which will change, as developers gain more experience with Bluetooth LE Audio, but it will take time.

Even where developers are starting from scratch with a Bluetooth LE Audio only design, there are still real-world resource issues to address. The new Core and Generic Audio specifications describe a perfect world, where devices always have sufficient resources to do everything when it is needed. Implementers often need to make compromises to meet price and time-to-market expectations. A common example is the everyday stereo application, where the specifications imply that there is a separate LC3 instance for each audio stream and sampling rate, allowing everything to happen in parallel. In practice, that may not be the case and developers need to think about how serial encoding of input audio streams affects latency.

In this real world there are trade-offs, and life may not be quite as simple as many of the diagrams in the specifications and this book suggest. This chapter examines some of the issues that have been seen during early Bluetooth LE Audio development and testing. Most of the principles have been covered in earlier sections, but it is useful to highlight some of the common misconceptions and implementation issues that have been seen, as well as providing a few examples of how specific design features may be addressed.

13.1 Latency and Presentation Delay in real implementations

Bluetooth LE Audio has been designed to support very low latencies. That tends to ignore real-world constraints, which limit how many processes a device can perform concurrently. There is also an issue that developers follow recommendations for Presentation Delay without thinking about how they can be optimised. In this section we look at how those affect performance.

13.1.1 Single LC3 instances

All of the diagrams in the Bluetooth LE Audio specifications which deal with stereo streams assume that transmitters have multiple LC3 instances, so that encoded SDUs are available for both left and right streams (and different sampling frequencies) at the same time. In practice, that may not be the case. There’s also an assumption that Acceptors wait until the Synchronization Reference Point before they do any decoding, which is also not necessarily true.

If we follow the diagrams in the specifications, they result in a picture for the overall latency which looks like Figure 13.1. The x-axis timings are approximate, with the exception of the capture timing, which is set by the 10ms sample frame. The other timings will depend on the efficiency of the platform for each of the tasks. The Rendering Point indicates the start of rendering. Decoded samples are rendered at the same rate at which they were captured to produce a continuous output stream.

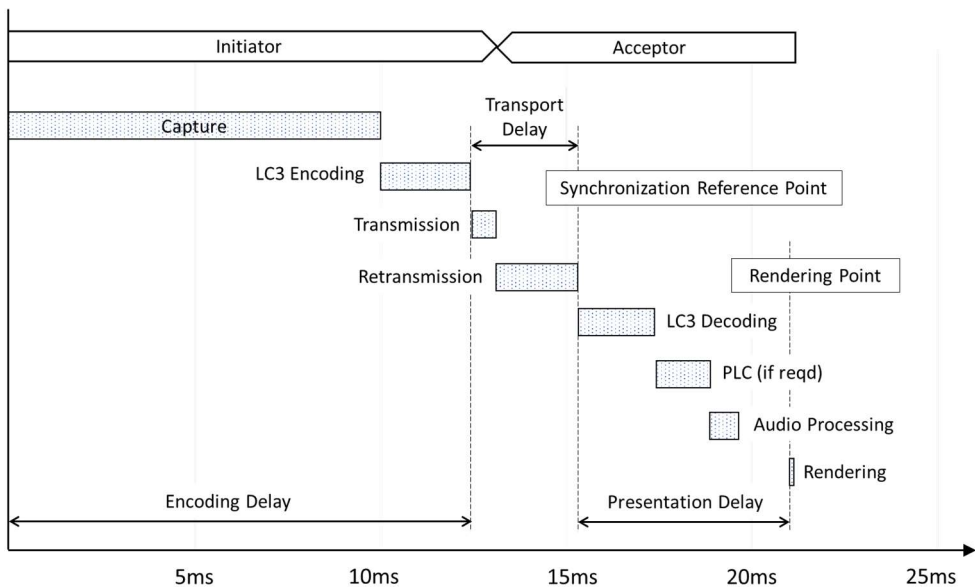


Figure 13.1 “Ideal” latency diagram for broadcast

The implication is that for multiple input streams everything is done in parallel and everything is optimised, so that SDUs are available for transmission and decoding “just in time”. With those assumptions, it’s possible to achieve an end-to end latency of just over 20ms, assuming that the receiver can support a Presentation Delay of around 7ms. With a limit of two retransmissions per channel and a 5ms Presentation Delay, it should even be possible to get down to around 18ms. That’s the theory, but the practice is often different, not least because many implementations adhere to the Presentation Delay value of 40ms, which is defined as mandatory for a Broadcast Sink to include in BAP. It is not mandatory for a Broadcast Source to use that value – it can choose any value. However, as the 40ms is the only value mentioned

Section 13.1 - Latency and Presentation Delay in real implementations

in the specifications, many implementations adopt it without thinking why, resulting in an overall latency considerably higher than is optimal for the use case. Having said which, implementers need to be aware of what a Broadcast Transmitter's intended Broadcast Sinks are able to support.

Figure 13.2 brings us a little closer to reality. Here the transmitter has limited resources and can only support one LC3 encoding instance. Although it can (and must) capture both of the left and right input streams in parallel, it is shown encoding the left capture first, followed by the right capture. It is unlikely that it would start transmission until both SDUs are ready to be sent in BIS1 and BIS2 payloads. Compared to the "ideal" situation of Figure 13.1, this introduces a new element to the overall latency.

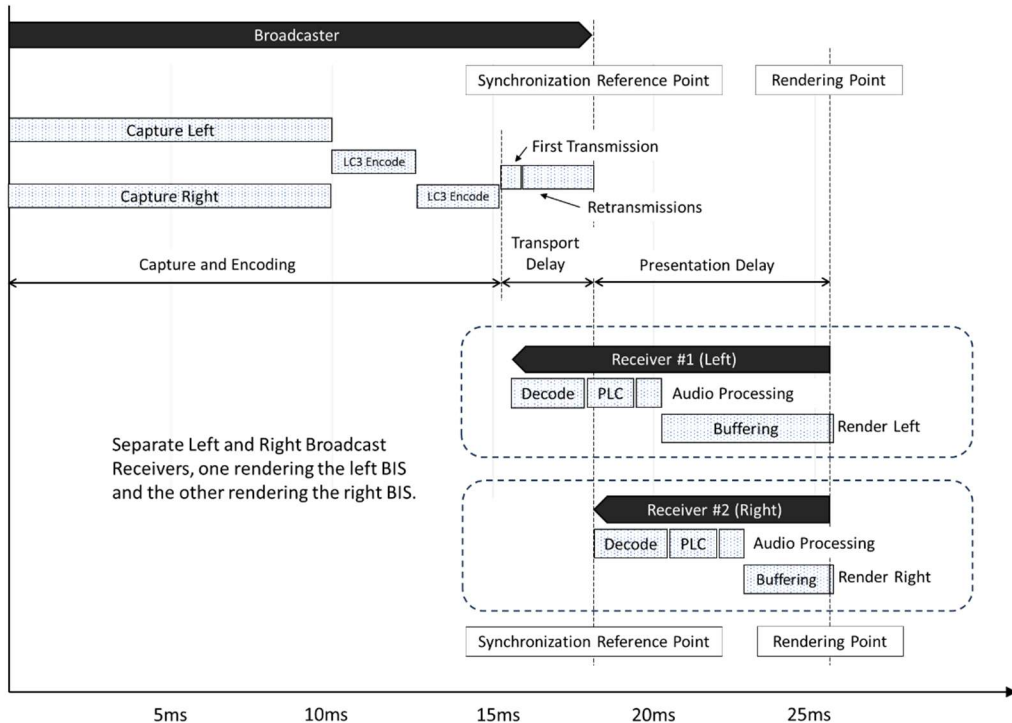


Figure 13.2 A real-world example of a broadcast transmission

On the reception side, Figure 13.2 has two separate earbuds, each of which can operate independently, as they share no resources. The diagram shows Receiver #1 successfully acquiring a packet at the first transmission. It can immediately start to decode it – that would be the natural design option for most firmware engineers. Once decoded, packet loss correction may need to be applied, followed by any application specific audio processing. The resulting audio sample then needs to be buffered before rendering starts at the Rendering Point defined by the Presentation Delay.

Receiver #2 in this example experiences interference, so doesn't acquire a good packet until the final retransmission. As a result, the last received octet of its received data packet coincides with the Synchronization Reference Point. This means Receiver #2 needs to buffer the

decoded data audio for less time before rendering commences – also at the Rendering Point. The set of tasks it needs to perform between the Synchronization Reference Point and the Rendering Point defines the minimum Presentation Delay which it can support. This must also apply to every other member of the Coordinated Set, as all set members must support a value which allows the last device to receive its last packet within the BIG with enough time remaining to perform all of its processing. As most pairs of earbuds and hearing aids are supplied as pairs, this is normally determined and set by the manufacturer. For each Broadcast Receiver, you can't predict which BIS subevent will be successfully received. It could be the first transmissions, or any of the following retransmissions. Hence the buffering time needs to be dynamic, potentially changing in each Isochronous Interval.

It's instructive to look further with the example of Figure 13.3. The difference here is that the Broadcast Receiver is a pair of headphones, which is receiving both of the BISes from the Broadcast Transmitter. Because they are resource constrained, they have the same issue as the Broadcast Transmitter of having to run their single LC3 decode instance serially on the two received Audio Streams.

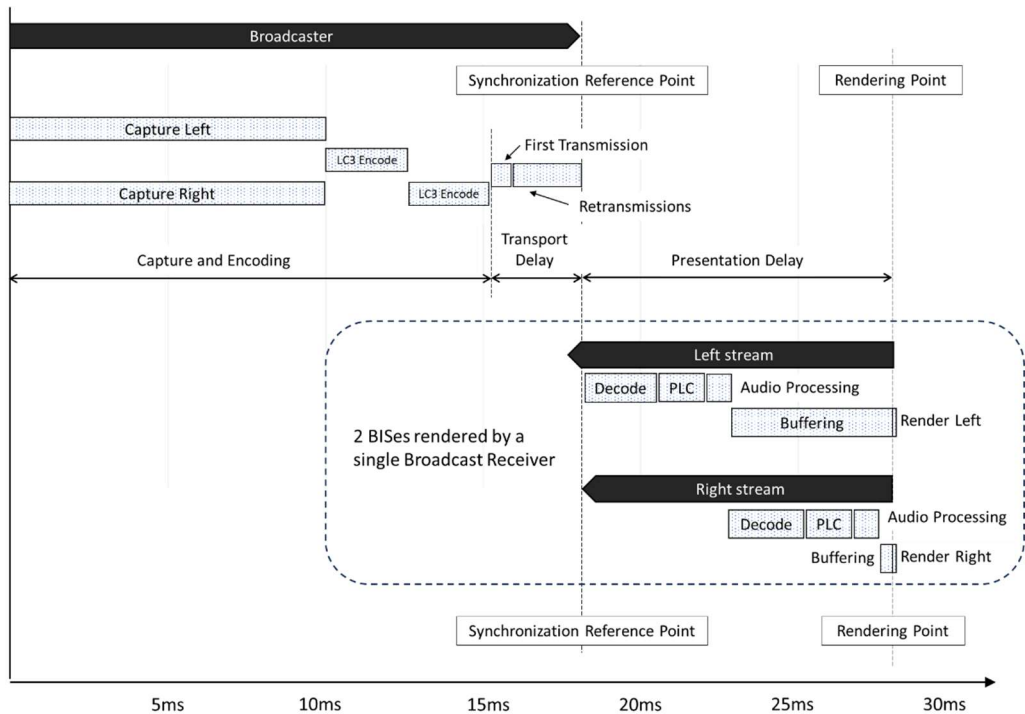


Figure 13.3 A real-world example with a Broadcast Receiver rendering two audio streams

In this case, the Broadcast Receiver fails to acquire the packets for left and right BISes on their initial transmissions, only receiving them on their final retransmission. The left packet is acquired one subevent before the Synchronization Reference Point, while the right packet is the last transmitted packet, in the subevent ending at the Synchronization Reference Point. The left stream is processed as soon as it is acquired, but the right stream cannot be processed until the DSP has completed the LC3 decode and/or PLC for the left stream, delaying it for

Section 13.1 - Latency and Presentation Delay in real implementations

around 5ms. This means that the minimum value for Presentation Delay in the headphones is greater than the individual earbuds of the previous example, as it has to allow for serial processing of the contents of the two BISes. It is worth noting that headphones with this resource constraint may have a higher limit on their minimum Presentation Delay than similar earbuds.

Two points should be noted from these examples. The first is that the real world is unlikely to be as efficient as the diagrams in the Bluetooth LE Audio specifications may lead you to believe. To get close to the maximum theoretical performance may require a lot of resource and optimisation. In practice, designers need to look at the trade-offs and decide what is relevant for each application. The second point is that neither the Broadcast Transmitter, nor the Broadcast Receiver are aware of the resource constraints of the other. Broadcast Receivers can be designed to reduce the combined LC3 decode and PLC duration to around 3ms, with just a few more milliseconds for local audio processing. Other implementations may struggle to complete it within 20 ms. Broadcast Transmitters are unaware of this, which is why the Auracast™ Simple Transmitter Best Practices Guide suggests using a Presentation Delay value of 40ms, which all Broadcast Receivers can support. However, this approach does result in overall latencies that are typically greater than 60ms.

This is the reason that the BAIRF (Broadcast Audio Immediate Rendering Flag) LTV was introduced, to indicate to Broadcast Receivers that they can implement their own choice of Presentation Delay, based on their internal knowledge of their minimum supported Presentation Delay value. The Broadcast Transmitter would generally include this LTV in the metadata section of its Public Broadcast Announcement whenever its use case is associated with a low latency, typically accompanied with a Context Type of «Live».

The BAIRF LTV is only applicable to broadcast. Unicast has the same issues, but during ASE configuration, devices can expose their maximum, minimum and preferred Presentation Delay values, allowing the Initiator to optimise the system. Although this gives the Initiator knowledge of latency constraints in its Acceptors, the Acceptors have no way of knowing about any delays introduced by the Initiator, so the overall Audio Latency is not exposed.

13.1.2 Audio Processing at the Broadcast Source

Figure 13.4 shows one further real-world element which may be present in a Broadcast Source, which is audio processing.

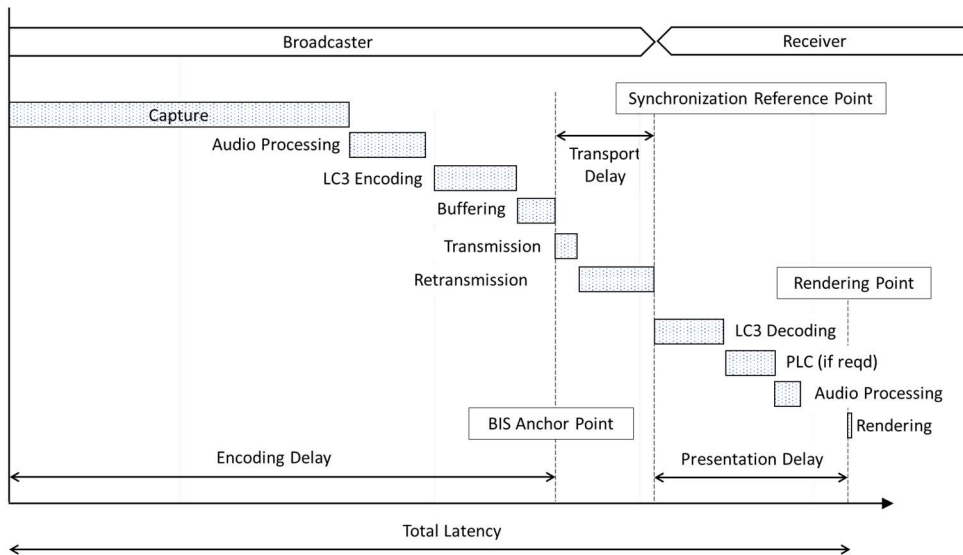


Figure 13.4 The addition of Audio Processing in a Broadcast Source

Audio Processing is typically performed after the audio capture, and may be performed for various reasons, such as:

- Creating a mono channel from a stereo input. This could be transmitted either instead of, or in addition to the stereo input channels.
- Increasing intelligibility by preprocessing the input audio to enhance the voice portion of the stream. Such an enhanced stream would normally be tagged as an ALS (Assisted Listening Stream).

The only effect of the extra audio processing is to increase the time before the captured audio is encoded and transmitted. Applications requiring the lowest latency should be aware of it and how it may affect their use cases.

13.1.3 Assisted Listening Streams (ALS)

Assisted Listening Streams are intended to allow users with minor hearing loss to identify a pre-processed stream that they can use with standard earbuds. An example is the “Dialogue Boost” options on streaming TV services. Many cinemas and theatres also provide streams which have been processed to enhance audibility. The metadata LTV alerts users to its presence. These streams may be pre-processed by the audio content provider, in which case they don’t introduce any additional latency, or they can be generated dynamically. The latter is the case for live performances, or may be performed by an audio algorithm in an Auracast™ transmitter, in which case it would increase the latency, as is the case with the example in Figure 13.4.

Section 13.2 - Mixing Broadcast and Unicast

The ALS LTV also allows hearing aids to identify the presence of a processed stream, as they may prefer the non-processed audio. This is the case where they perform their own audio processing which has been customized to the wearer. These devices typically prefer to use the original non-ALS audio stream, and use their own processing after applying the Packet Loss Correction stage, as shown at the right of Figure 13.4.

13.1.4 Presentation Delay for unicast applications

Unicast applications have far more control over the end-to-end latency, as an Initiator receives values of Maximum, Minimum and preferred Presentation Delay during the codec configuration operation. These are defined in Table 4.3 of ASCS. Each Acceptor returns values for its

- Presentation Delay (Min)
- Presentation Delay (Max)
- Preferred Presentation Delay (Min)
- Preferred Presentation Delay (Max)

The preferred values indicate the preferred working range and cannot fall outside the Presentation Delay (Min) and (Max) values. They are expressed for each ASE, and can take the same value, which can be identical to the Presentation Delay (Min). Implementers should beware that setting them to the same value may cause issues for an Initiator creating a CIG comprising Acceptors from different manufacturers or which have different firmware versions, as a lack of overlap would mean there is no common value. In general, the Preferred Presentation Delay (Min) can safely be set to the Presentation Delay (Min) value.

Initiators should choose a value which all of their Acceptors can support for the same ASE direction, generally setting it to the lowest common value, unless the application has a specific reason for a higher value.

13.2 Mixing Broadcast and Unicast

Many Broadcast Transmitters will never need to support unicast, but confine themselves to transmitting broadcast audio streams, using the configuration they were initially set up with. However, a growing number of applications are expected to emerge for personal devices, such as phones, tablets and PCs, which can be switched from sending a personal unicast stream to sharing a broadcast stream. The use case driving this is the new ability of Bluetooth LE Audio to switch between a personal audio stream, and personal broadcast streams which can be shared with friends, family members and colleagues.

Although it seems a simple concept, a number of manufacturers have struggled to produce interoperable implementations. As the transition between unicast and broadcast exercises many elements of the specification, it is a good candidate to step through in detail, as it helps to illustrate the way the different specifications come together. In the preceding chapters we've focused predominantly on unicast and broadcast in isolation. However, the overarching

goal of Bluetooth LE Audio is for them to work as a single Bluetooth LE Audio ecosystem. When they come together, it's important to consider the user interactions, ensuring they are designed to make the process simple.

To illustrate this, Figure 13.5 shows three friends – Tom, Dick and Harry, who are respectively wearing earbuds, hearing aids and a pair of headphones to listen to a TV, which is broadcasting a stereo audio stream. At some point, Tom's phone rings and he takes the call, dropping the broadcast audio stream from the TV. During the call, he receives a link for a new video which he decides to share with his friends after he has finished his call.

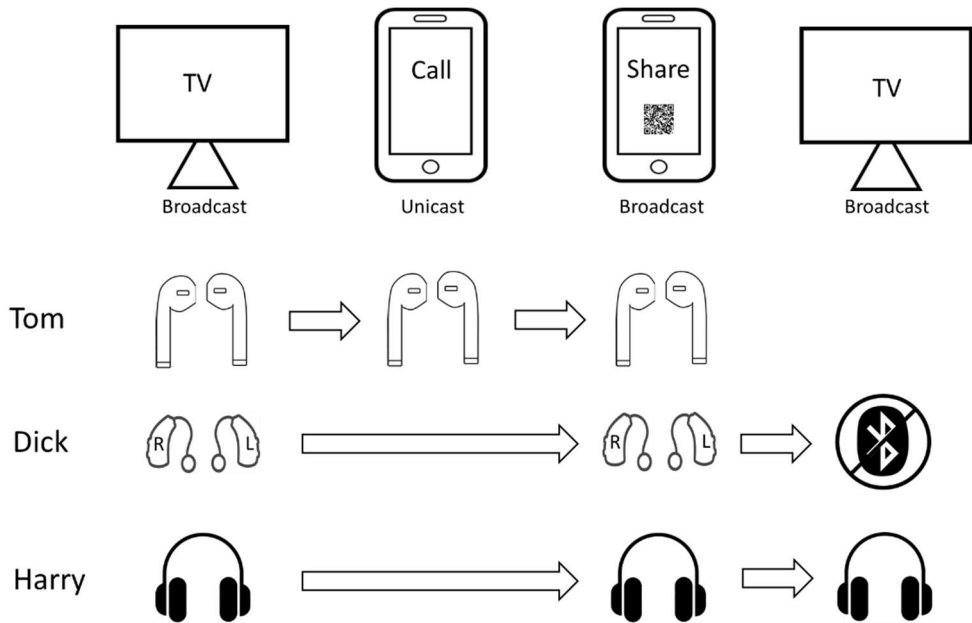


Figure 13.5 An example of broadcast and unicast transitions

To share the video's audio stream, Tom needs to convert his current unicast link to broadcast, and share information with his friends, so that they can move from the TV broadcast to listen to his phone. Both Dick and Harry join Tom's broadcast.

At the end of listening to Tom's video, Dick decides to turn off the Bluetooth link on his hearing aids, so that he can have a conversation with another friend. Harry goes back to watching the TV.

The methods for effecting the change from personal to shared audio and vice versa are documented in CAP and BAP. The transitions involves terminating a unicast stream prior to establishing a broadcast stream carrying the same audio content and making it available for authorised users. To return to receiving a broadcast is the reverse – terminating the unicast

Section 13.2 - Mixing Broadcast and Unicast

stream and scanning to find a Broadcast Source.

To provide a good user experience, devices supporting this transition need to provide a clear user interface to make it easy to perform. If Auracast™ support is claimed, this includes a number of options or defaults which should be presented to the user to ensure that the resulting broadcast stream meets the requirements of the Auracast™ Simple Transmitter Best Practices Guide, to ensure that it is compatible with Dick’s hearing aids. We can now walk through the scenario of Figure 13.5 to see how this can be accomplished in a user-friendly way.

13.2.1 The transition from broadcast to unicast

The first transition is when Tom receives his phone call. When he started listening to the TV broadcast, his earbuds will probably have used CCP to instruct his phone to use out-of-band ringtones. This means that the phone had no need to establish an isochronous channel with his earbuds before Tom makes a decision about accepting or rejecting the call. In our example, Tom accepts the call, which he can do on either the phone or his earbuds. This results in “Answer Incoming Call” being written to the TBS server on the phone. The phone will notify this, so that the other earbud is aware of what’s happening, at which point both of his earbuds will terminate their synchronization to the BIG, and the phone will start to configure their ASEs to set up a bidirectional CIG for his phone call. As far as Dick and Harry are concerned, nothing has changed. They carry on listening to the TV’s broadcast Audio Stream.

13.2.2 The transition from Unicast to Broadcast

At the point that Tom finished his call and tells Dick and Harry that: “You need to listen to this”, things start to change. The app that Tom is using for the video stream has an “Auracast™ Sharing” option, which converts his unicast stream to a broadcast. Figure 13.6 illustrates the basic change in topology that occurs when Tom makes that decision to move from unicast to broadcast.

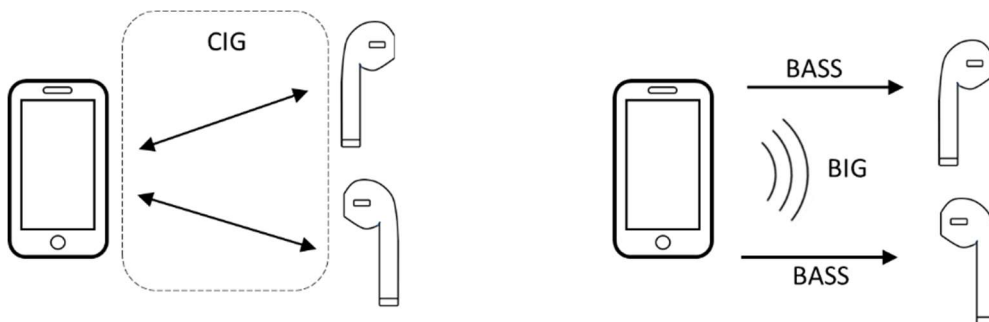


Figure 13.6 The transition from unicast to broadcast

In Figure 13.6, Tom starts with the topology on the left of the diagram, where the phone has established a CIG comprising two CISes to stream audio to his pair of earbuds. After he

presses “Auracast™ Sharing” to move to broadcast, the CIG is terminated and the phone sets up a BIG, which generally contains the same number of audio channels as were in the CIG. The phone then uses the Modify Source operation⁸⁰ specified in BASS to write to the Broadcast Audio Scan Control Point characteristic in each of the earbuds, instructing them to synchronise to the appropriate BIS within the BIG. At the end of this process, Tom’s earbuds will be rendering the same audio content as they were when the phone was acting as a unicast Audio Source.

It would be more correct to say “the same input audio content”. As part of the transition, the phone app may have decided to change the QoS setting that it used for the unicast phone call. This may be by default, the result of a previous configuration choice Tom made, or a specific option that appeared when Tom pressed “Share”. The reason for flagging this up is that Dick is wearing hearing aids, which are unlikely to be able to accept any QoS setting with a sampling rate higher than 24kHz. If Tom’s phone claims to be Auracast™ compliant, it must make this choice available to Tom to allow hearing aid users to access the resultant broadcast audio stream.

For Tom, there is likely to be a short gap between the termination of the unicast audio stream and the resumption of the stream after his earbuds have been instructed to synchronise to the new broadcast. This can normally be “hidden” by presenting information to the user, notifying them that their broadcast stream is being set up. At the UI level, there may be decisions which the user needs to make (which may be skipped if appropriate defaults have been set), which can also mask the small interruption to the audio stream. We’ll look at those in more detail in Section 13.2.6.

As soon as Tom hears the audio return to his headphones, he can tell Dick and Harry that they can share it. Both find it in the same way they found the TV, using their Broadcast Assistants to locate Tom’s phone and selecting them as their new Broadcast Source. In 13.2.5 we’ll look at ways of making that even easier.

Throughout this process, the control functionality, such as volume, balance and muting is unaffected, as this is controlled by VCP, which operates on the control plane and is independent of the form of the audio stream. After the transition from unicast to broadcast, Tom can continue to set his personal rendering preferences, speeding up, pausing or stopping the playback, as well as making decisions about further transitions, such as transferring to a phone call. Any future audio transition that Tom initiates is likely to terminate the broadcast transmission, as most Bluetooth LE Audio Sources do not have sufficient resources to support

⁸⁰ As Tom’s earbuds are paired with his phone, the phone will probably have added itself as a Broadcast Receive Source characteristic instance by issuing an Add Source operation when they connected. So, in this instance, the phone issues a Modify Source command to that existing instance. It will normally also issue the Set_Broadcast_Code command to transfer the Broadcast_Code.

Section 13.2 - Mixing Broadcast and Unicast

concurrent unicast and broadcast streams.

Dick and Harry can also control their own personal volume independently of Tom and each other, as their volume control apps work directly with their headphones and hearing aids. At any point they can choose to transition to another broadcast source or phone call. The only thing they can't do is control the media playback on Tom's phone, as only he has access to his Media Control Server.

13.2.3 Leaving and choosing other broadcast streams

In the example above, both Dick and Harry decide to move on after having listened to Tom's video. Both would use their phone to accomplish this. Dick goes to his hearing aid app and chooses to stop listening to the Bluetooth audio stream. This would terminate synchronization with Tom's broadcast audio stream, but would maintain the ACL connection, allowing him to adjust his hearing aid's volume and other hearing specific functions. At any point, he could look for another Broadcast Source, or connect to his phone or another device to stream music or make a call.

Harry decides to return to the previous TV broadcast. He can use his Broadcast Assistant to do this, or could make use of a "reconnect to last broadcast" function, where his earbuds would remember the details of his last session and resynchronize by using the information in the most recently used Broadcast Receive State characteristic. That is an implementation choice from the earbud manufacturer, but would enhance the user experience.

In the description above, Tom has not appeared to use a Broadcast Assistant. That's because his smartphone contains both a Broadcast Source and a collocated Broadcast Assistant. As soon as the Broadcast Source starts operating, it instructs that Broadcast Assistant to write to his earbuds, without any further user interaction with Tom. All he had to do was select the "Auracast™ Share" option.

13.2.4 Controlling the transition

Having covered the basics of the use case, we can now look into a bit more of the underlying details. The handover process is defined in CAP's Unicast to Broadcast Handover procedure [CAP 7.3.1.10] and the Broadcast to Unicast Handover procedure [CAP 7.3.1.11], but quite a lot of that is about handling Context Types. It's useful to look at details of the transition. The figure below shows the key stages of the process of handover:

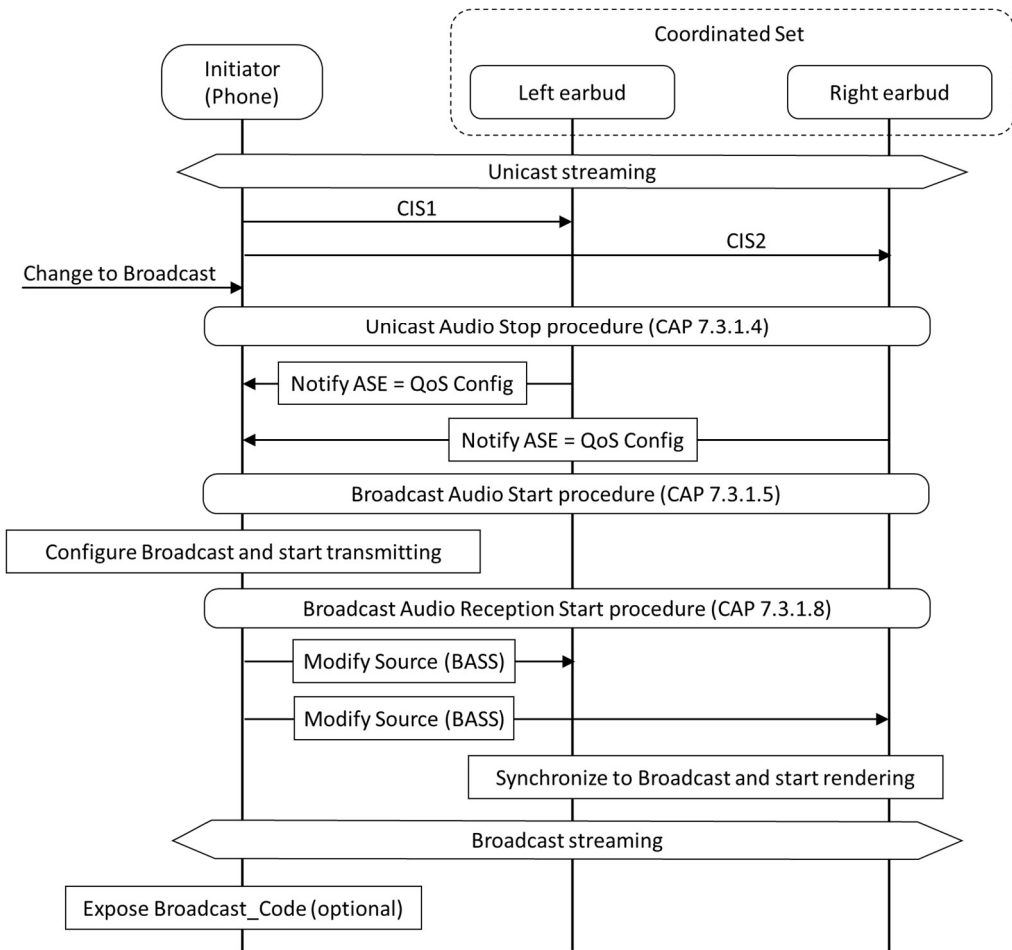


Figure 13.7 Key stages in the transition from unicast to broadcast

The process starts with the Audio Source (the phone) streaming audio to both earbuds, which are members of a Coordinated Set. That is important from a CAP perspective, as it means that anything which the Initiator does must be applied to both members of that set, which are the left and right earbuds. The audio is carried over CIS1 and CIS2, which will contain the left and right audio channels respectively.

Unicast streaming continues until Tom makes a selection to change to broadcast. In this case it's to share audio with his friends, but it could be to send the audio stream from his phone to a set of multiple home speakers which are configured to receive and render broadcasts from his phone.

At this point, the Initiator needs to tell each of its earbuds to terminate unicast streaming by disabling their ASEs. It is up to the implementation whether it takes them to the QoS enabled state or releases them fully, but in most cases it is recommended that it is the former, as this will allow a faster transition when the phone wants to transition them back to unicast operation.

Section 13.2 - Mixing Broadcast and Unicast

When each earbud has returned its ASE state to the QoS configured or Disabled state, the Initiator can proceed to set itself up as a Broadcast Source using the same input audio stream. The procedure is defined in CAP as the Broadcast Audio Start Procedure. If the Initiator claims compliance to the Auracast™ requirements, it needs to determine whether it can use the same codec configurations that it had used for its unicast stream. This may be presented as a user interface choice, or may be a factory default or user configuration. As a result of the decision, the Initiator may transmit multiple BISes in one or more BIGs. These options are explained below.

When the stream format decisions have been made, the Initiator should start broadcasting. As it still has an ACL link with each of its earbuds, it can instruct them separately to synchronise to the appropriate BIS by writing a Modify Source operation to their Broadcast Audio Scan Control Point characteristics. This will include information on which specific BIS each earbud should use. If the Initiator supports PAST, it should use this to decrease the time for synchronisation, as well as removing the need for the earbuds to scan.

If the streams are encrypted, the Set Broadcast_Code operation should be used to securely send the Broadcast_Code to each of Tom's earbuds, unless they have stored it from a previous session and that earlier value is still valid.

Once each earbud has synchronised to its intended broadcast stream it will notify the PA_Sync_State parameter of its Broadcast Receive State characteristic instance, confirming to the Initiator that it is now rendering the broadcast stream.

Once both earbuds have notified their PA_Sync_State as “Synchronized”, the transition from unicast to broadcast is complete. At this point, Tom's phone can let Dick and Harry know about the availability of the new broadcast stream, so that they can acquire it. Tom's phone will already be advertising it in its extended advertising, but we're implying a notification on the screen of Tom's phone as well.

13.2.5 Announcing the availability of the broadcast stream to other users

The most common use case anticipated for the unicast to broadcast transition is sharing audio from a mobile phone, tablet, PC or TV. Multiple users can easily find these broadcasts using a Broadcast Assistant, but if they are personal streams, they are likely to be encrypted, which requires each user to obtain the relevant Broadcast_Code. Within the Bluetooth LE Audio specifications, there is no mechanism to transfer this, which can result in a clumsy requirement, like the common Wi-Fi experience, of a user having to key a code into an application.

In Figure 13.7, the final operation of “Expose Broadcast_Code” is shown as optional. If the streams are not encrypted, there is no need for a Broadcast_Code. If the streams are encrypted, the phone application needs to determine how to expose the Broadcast_Code. For most personal applications, this method of distribution should usually be private. Any device which is scanning will be able to tell that the broadcast is happening, but will not have access to the Broadcast_Code to decrypt it.

To improve the user experience, the Bluetooth SIG has published the Broadcast Audio URI specification, which can be used to simplify the user experience by allowing all of this data to be displayed in a QR code on the broadcast device, which another user can scan with their own phone. Figure 13.8 shows how this can be used.

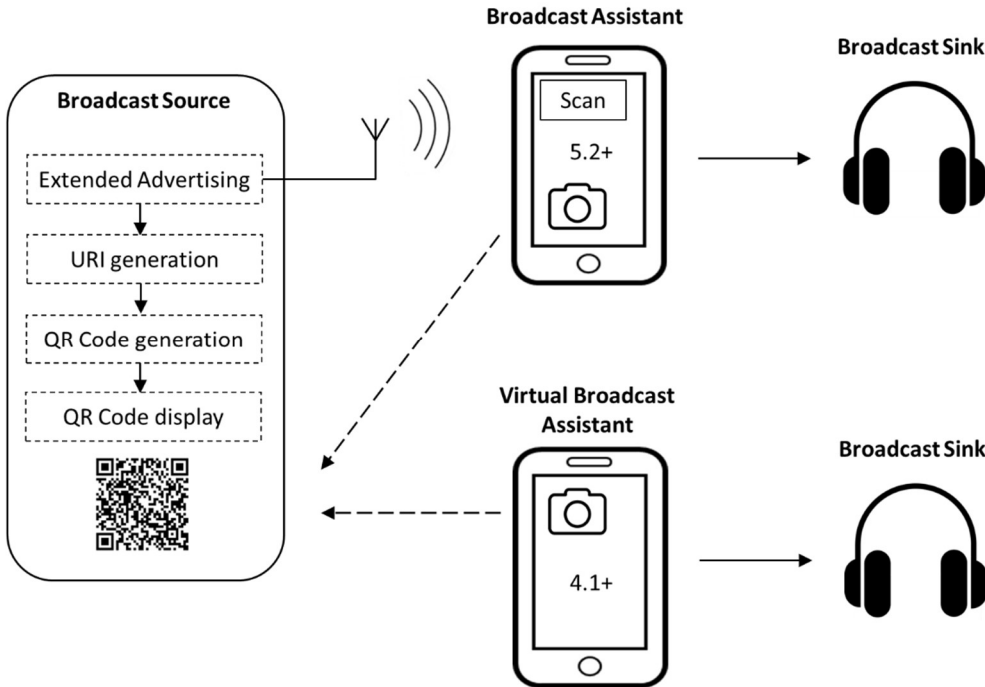


Figure 13.8 Alternative methods to connect to a broadcast stream

On the left, the Broadcast Source is shown generating extended advertisements, which allow any scanning device, such as the Broadcast Assistant shown at the top to discover the broadcast. These advertisements contain all of the information that a Broadcast Assistant requires to present the available options to a user and then request that their Broadcast Sinks synchronise to their preferred broadcast. If there are multiple broadcasts within range, the user needs to determine which is the correct one. This solution requires a Broadcast Assistant that complies with Bluetooth® Core 5.2 or above, in order to support scanning for extended advertisement. The user will also need to independently discover the Broadcast_Code if the stream is encrypted.

If the Broadcast Source supports the Broadcast Audio URI, it can encapsulate all of the information that is contained in the extended advertisements into a single URI string, which can be displayed as a QR code. This could be a printed code next to the transmitter, or in the case of Tom's phone, it could be dynamically generated on the screen. An app on either smartphone in Figure 13.8 can scan this QR code and extract the same information to instruct its connected Broadcast Sinks to synchronise to the broadcast streams. It's a very simple and

Section 13.2 - Mixing Broadcast and Unicast

attractive user experience. A further advantage of using the QR code is that it encapsulates the Broadcast_Code for an encrypted stream, so the person scanning the QR code does not have to enter any more information – their headphones, hearing aids or earbuds will just connect to the audio.

Examples of how the QR code might be displayed are shown in Figure 13.9, where it is shown being generated and displayed on a smartphone. When supporting audio sharing, apps should also provide basic sharing information for users who cannot scan QR codes, such as providing a Passcode, which could be entered into a scanning application.



Figure 13.9 Using a QR Code and basic scanning information to share a personal broadcast

A further, major advantage of using a QR code is that it can be supported by any device that has a camera and supports Bluetooth LE, going back to version 4.1. Whereas a scanning Broadcast Assistant requires support for Bluetooth® Core 5.2 or above, which generally needs a user to have a new phone, the “virtual Broadcast Assistant” approach can be implemented on most phones which have been shipped in the last ten years. Details of how to design a virtual Broadcast Assistant are contained in the Bluetooth SIG document “Developing Auracast™ Receivers with an Assistant Application for Legacy Smartphones”. Information and examples of the use of QR Codes is included in the Broadcast Audio URI specification.

Once you know that your friends are receiving your audio stream, you can instruct your phone to stop transmitting the advertising set, so that nobody else is aware of your broadcasts. If other friends want to join in, you just re-enable it.

13.2.6 Selecting codec configurations when transitioning from unicast to broadcast

CAP, along with most of the preceding discussion, makes no comments on the processing of the audio stream, giving the impression that the transition from unicast to broadcast continues to use the same QoS parameters. In many cases, that may be true, but if the Broadcast Transmitter claims Auracast™ compliance, there are caveats that may require a change of codec setting before the broadcast streams are established.

The Auracast™ Simple Transmitter Best Practices Guide mandates that all Auracast™ Transmitters must support QoS settings with LC3 sampling frequencies of 16kHz and 24kHz. As every Auracast™ Receiver is capable of decoding these settings, this ensures that there is universal compatibility. However, an Auracast™ Transmitter may also use the LC3 codec at a 48kHz sampling frequency, to provide a higher perceived audio quality. There is a condition attached to this, which is that an Auracast™ compliant Broadcast Source must include a user accessible option to change the encoding from 48kHz to 24kHz or 16kHz on request. This allows users with hearing aids to ask that an accessible Auracast™ transmission be made available for them.

The most common time where it is anticipated that this request will be made is when somebody like Tom decides to share their audio stream from a phone or TV. Hence it makes sense that when an app is asked to transition from unicast to broadcast on a device which is currently streaming at 48kHz, the user is asked whether they want to continue at 48kHz, or reconfigure to 24kHz. (In general, 16kHz is only appropriate for voice and is not a preferred option for music, which should use 24kHz.) The resulting choice will then be applied to the configuration of the broadcast Stream.

The choice for the QoS configuration of the broadcast streams at this point is principally driven by the airtime requirements of the various options. Table 13.1 shows the common combinations of the 48_2_2 and 24_2_2 QoS configurations that provide compliant Auracast™ broadcast streams and the typical airtime usage for each of them.

- The top two rows show the normal settings for Personal Auracast™ transmissions, assuming High Reliability QoS settings.
- The middle two rows do the same for Public Auracast™ transmissions. For live applications, these would probably use the Low Latency options, reducing the airtime to 13% or 26%.
- The final four rows demonstrate combinations of QoS settings which fulfil the requirements for both Personal and Public Auracast™ transmissions simultaneously.

Section 13.2 - Mixing Broadcast and Unicast

	Number of BIGs	Number of BISes	48_2_2		24_2_2		Total Airtime
			Mono	Stereo	Mono	Stereo	
Personal	1	1	30%				30%
	1	2		59%			59%
Public	1	1			22%		22%
	1	2				43%	43%
Accessible	1	3		59%	30%		89%
	1	2	30%		30%		60%
	2	2	30%		22%		52%
	2	3		59%	22%		81%

Table 13.1 Auracast™ Compliant options and airtime usage for devices supporting 48kHz QoS

As can be seen in Table 13.1, splitting the 48_2_2 and 24_2_2 streams into separate BIGs can save some airtime. Where different QoS settings are used in the same BIG, then airtime is wasted, because every BIG subevent is sized to cope with the largest packet.

Implementations may also consider reducing the number of retransmissions, although this needs to be balanced against the potential loss of robustness. They may also consider transmitting mono streams at both the 24kHz and 48kHz sampling rates. The penultimate (shaded) row Table 13.1 shows that this is the most attractive option in terms of airtime usage. Subjective tests with different combinations have shown that most users cannot distinguish any difference between LC3 encoded audio sampled at 24kHz or 48kHz in an environment with normal levels of background noise present. Users have also rated mono LC3 streams as “excellent”. Developers need to decide whether the ability to share audio, or transmitting audio at the highest possible quality, (which may not be interoperable with all devices), is more important in driving the new user experience and market adoption of broadcast audio.

Reiterating the Auracast™ requirements, if a Broadcast Source wants to support operation at a 48kHz sampling frequency and also claim Auracast™ compliance, it must be able to transition to one of the six options in the highlighted “Public” and “Accessible” rows of Table 13.1.

13.2.7 Advanced considerations

Some Initiators may be capable of transmitting both unicast and broadcast audio streams concurrently, but it is not expected to be common. For such a device, there would be no need for the transition described above – the unicast transmission would be supplemented by a complementary broadcast. In this case, the broadcast could continue at a point where the user transitioned their earbuds back to unicast, which could be because they wanted to take an incoming phone call. However, this behaviour is not expected to be supported by the majority of phones or other Bluetooth LE Audio transmitters in the short term, because of the resource overhead involved.

13.3 Using broadcast with TVs

The TV industry probably has the steepest learning curve of any industry with Bluetooth LE Audio. That's largely because it has done its best to ignore the problems which came with streaming and Bluetooth Classic Audio, which are now coming home to roost. Whilst the smartphone ecosystem has managed to maintain a fairly good user experience, largely by confining virtually all interactions to the phone, TVs have had to cope with a proliferation of peripherals from remote controls and soundbars, to multiple external video sources, which include DVD players, set top boxes, video dongles and smartphone casting. All of these lead to consumer confusion, particularly over volume. Bluetooth LE Audio provides tools which can help solve some of those issues and offer a more consistent user interface, but they require a lot of existing practices to be reviewed.

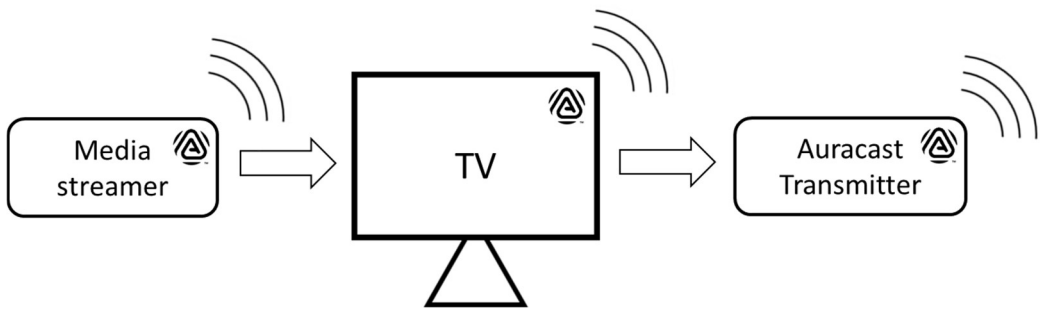


Figure 13.10 Broadcast Transmission options for a home TV

Figure 13.10 illustrates three different ways that a user may first experience Auracast™ audio streams with a home TV.

- On the left is a Media Streamer (TV Stick), which provides a video input from a streaming service. These are relatively low cost devices, which allow consumers to upgrade their video experience. The one shown here includes an Auracast™ Transmitter.
- In the middle is a TV which includes an integrated Auracast™ Transmitter.
- To the right is a dedicated Auracast™ Transmitter, which can be connected to one of the audio outputs of the TV.

Section 13.3 - Using broadcast with TVs

As TVs are relatively expensive, many consumers' first experience of Auracast™ will probably be with a Media Streamer or an external Auracast™ Transmitter. But over time they may end up with all three. Each of these can be transmitting an Auracast™ signal at the same time. The user may also have a soundbar connected to their TV, either wirelessly or via Bluetooth. There is enormous potential to confuse users, which means that the industry needs to think carefully about their user interfaces.

Although it may not be most people's first purchase, we should start with the TV by itself. It was one of the original use cases driving the development of Bluetooth LE Audio technology and demonstrates many of the challenges which the TV and video streaming industry need to address.

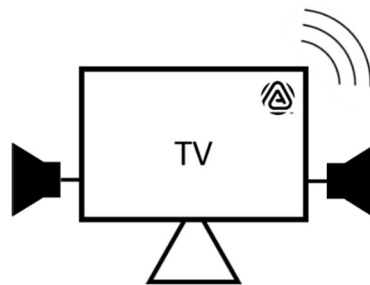


Figure 13.11 The simplest implementation of a TV with both ambient and Auracast™ audio

Figure 13.11 is a TV which is producing ambient audio through its internal speakers, as well as containing an Auracast™ transmitter broadcasting the same audio. It allows a family to watch the TV together, without someone with hearing loss constantly turning up the volume. Instead, they can listen with their hearing aid or earbud, setting a volume level for their personal comfort. This already challenges the orthodoxy of TV design in two separate ways:

- Both the ambient and Auracast™ streams are output at the same time. Most current TVs do not allow this option, but direct the audio output to only one choice, where ambient and external audio sources are not allowed to operate together.
- The volume of both audio sources are now controlled independently. The TV remote control can change the volume of the ambient output, as well as muting it, but the Auracast™ volume should be independent of the TV volume, and output at a “line” level to maximise the dynamic range for the LC3 encoder. With Auracast™, each user controls the volume of their personal device. Adjustments to the TV volume should not affect the transmitted Auracast™ signal. (The effect of muting the TV is not defined, but there is a strong argument that muting the TV's ambient audio should not affect the Auracast™ stream, which is already “silent” from the viewpoint of ambient sound.)

One of the reasons for the “either/or” of current designs is the relatively long latencies of A2DP, HDMI-ARC and optical audio output options, which are long enough to cause very

visible lip-synch problems. TVs can adjust their audio and video timings to compensate, but often don't know the delays being introduced, which can be several hundred milliseconds. If they were to provide an audio output at the same time as supporting a remote output it could generate echo, so they take the simplest approach of only supporting a single Bluetooth LE Audio output. With the low latencies available with Bluetooth LE Audio, that problem disappears. A TV can process its video stream to ensure that its local audio rendering is correct and feed that same signal to its Broadcast Transmitter and its analogue audio outputs. All three make the same audio stream available for devices to render without lip-synch problems. There may still be an issue with delays in audio sent via optical or HDMI-ARC outputs, which may involve delays both within the TV and the peripheral audio devices, but that is no different to what occurs today.

A second advantage of incorporating an Auracast™ Transmitter within the TV is that it can be used to connect both a soundbar and any number of hearing aids, earbuds and headphones, all of which will render at exactly the same time, as they share the same Presentation Delay.

The approach above solves one problem, but Bluetooth LE Audio's flexibility introduces another one, which is that users will expect to be able to choose from multiple Bluetooth LE Audio channels. Many streaming services include multiple Bluetooth LE Audio channels. As an example, Amazon's "Man in the High Castle" offers fifteen different options: English Dialogue Boost: Low, English Dialogue Boost: Medium, English [Audio Description], English, English Dialogue Boost: High, Deutsch, Čeština, Italiano, Español (Latinoamérica), Français, Polski, Magyar, Português, Español (España) and 日本語 (Japanese). As users become aware of this variety, they will expect their TV to support more than one language stream at the same time. That is particularly true of the Dialogue Boost options, which are of most use to earbud and headphone wearers with lower levels of hearing loss, and who have not yet started wearing hearing aids.

When a family wants to listen to two different languages at the same time, a different issue appears. Earbuds and hearing aids don't occlude all of the ambient sound. If the TV speakers are rendering the main language stream and the Auracast™ Transmitter is carrying a dialogue boosted stream it is not a problem for any listener. However, if the ambient stream is English and the Auracast™ stream is Korean, then the earbud wearer has to contend with hearing some of the English audio track at the same time. The practical solution is for all listeners to use earbuds or headphones and turn off the ambient sound. However, this means that the TV needs to broadcast two simultaneous Auracast™ streams. That's not an issue for Bluetooth LE Audio – it was designed to do that. However, it is a major change for TV manufacturers, who need to rearchitect their audio implementations to support transmission of multiple different language streams.

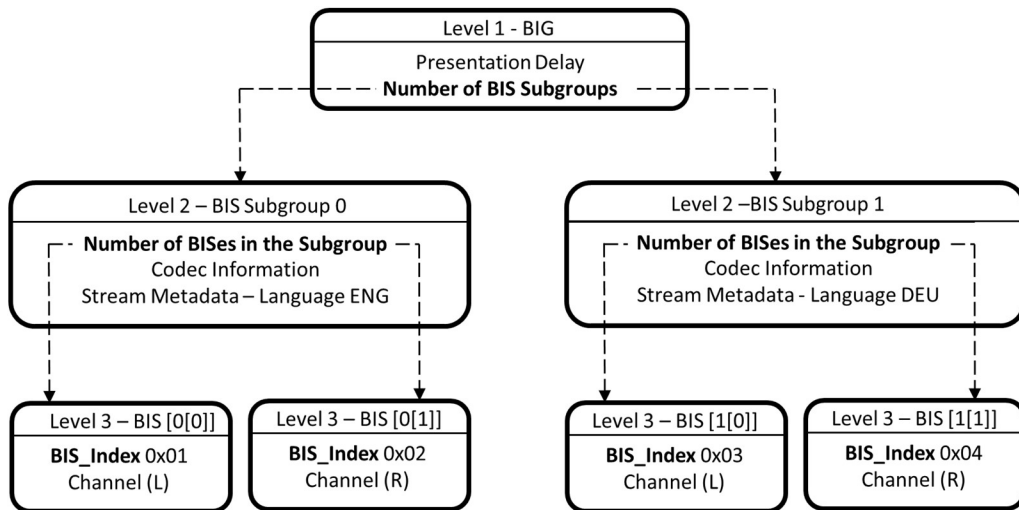


Figure 13.12 BASE for an Auracast™ TV transmitters sending two stereo channels

Figure 13.12 shows the BASE for a typical implementation, where a total of four BISes are transmitted. To ensure low latency and accessibility for hearing aids, it is recommended that they are configured with a 24kHz sampling rate and Low Latency QoS configurations. This allows all four channels to be accommodated with around 52% of airtime. As a streaming TV is likely to be using Wi-Fi to obtain the video stream, it is important to leave sufficient airtime, as the home Wi-Fi router may be set to use the same 2.4GHz band.

This approach requires major changes for TV designers and streaming services:

- The user menu must allow users to select more than one audio channel for Bluetooth LE Audio transmission. To make this accessible, it may require an additional “Language” button on the TV remote control, or a “Bluetooth button” which takes the user to the Broadcast Audio options, which would include multiple languages. In the shorter term, the menu should allow users to combine ambient sound with the same stream being broadcast.
- Streaming services need to support multiple languages in one stream. At the moment they don’t. Unlike multi-language DVDs, where every language is included on the disk, when you select a language with a streaming video services, the server provides a stream with only the audio track for that language. Another problem is that the language selection is contained within the streaming service app and is not exposed to the TV’s menu system. This means that it cannot currently be integrated into the TV menu or the Bluetooth LE Audio broadcast stream selection.
- When multi-language audio streams become available, TV video decoders need to be able to decode and route them to the appropriate output. That might mean:
 - Routing the same audio stream but providing both ambient and Bluetooth outputs,
 - Routing one audio stream to the ambient output and a different stream to

- the Bluetooth output, and
- Routing two different Bluetooth LE Audio streams to different subgroups in the same BIG.
- As the number of available language options varies with the programme, TV vendors and streaming services need to work together to extract the programme language for the TV user menu and implement defaults to be used when a particular language stream is not available.

These requirements involve a major change to the way that TVs are currently designed. It's also a major change for streaming providers, who do not currently include multiple language audio streams with their video stream. They also need to expose the language selection, or the dialogue boost options outside their own apps. Bluetooth LE Audio has enabled a new ability to share multiple Bluetooth LE Audio streams. The TV and streaming industry need to step up to support the possibilities, especially for listeners with hearing impairment.

Top end TVs, which is where Bluetooth LE Audio features will appear first, are likely to be expensive, so most consumers will not renew them until they break. To speed up access to these features, many users will look for them on Media streaming devices (also known as TV sticks or TV dongles).

13.3.1 The Auracast™ Media Streamer

A lot of innovation in TV content delivery in recent years has come from media streamers – the small dongles which you plug into your existing TV or display. These are likely to provide an early way for consumers to access Bluetooth LE Audio on their TV. These are distinct from Auracast™ Transmitters which plug into the existing audio outputs of a TV; they are the Roku and Chromecast type devices which typically receive a TV stream using Wi-Fi and connect to the HDMI input of a TV.

Figure 13.13 illustrates the main functional blocks of both an Auracast™ media streamer, and the TV it is attached to.

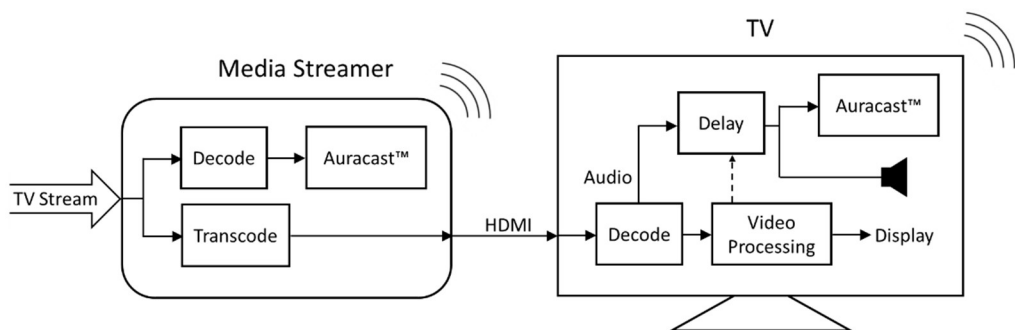


Figure 13.13 The main components of a TV streamer and Auracast™ TV

Starting with the streamer, it needs to convert the incoming signal into the correct form for its HDMI output. This is normally performed by transcoding the input signal. If the streamer

Section 13.3 - Using broadcast with TVs

provides a separate audio output, which in this case feeds its Auracast™ Transmitter, it needs to extract this stream, and decode it or transcode it into the LC3 format. The streamer's UI needs to expand its Settings menu to allow the Auracast™ transmissions to be configured. As with the example of the TV, the selection of language is currently in each streaming service app and is not exposed to allow the media streamer to apply a default setting other than the overall preferred language.

Moving on to the TV, it needs to decode the HDMI signal into its separate audio and video components. As TVs become more complex and move to 4K, 8K and beyond, the task of processing the video signal becomes more complex, particularly if additional enhancements, such as support for snow reduction, sharpness control and motion smoothing are enabled. These add a delay to the video rendering. To maintain lipsynch, the TV controller applies a compensating delay to the audio outputs. As long as the Auracast™ QoS configuration is for low latency, this delayed audio signal can be fed to the internal speakers in parallel to the Auracast™ Transmitter, giving an excellent user experience.

A problem can arise when the Media Streamer is being used to transmit the Auracast™ signal alongside the ambient speakers output of the TV. As Figure 13.14 shows, there may be an appreciable Bluetooth LE Audio delay between the Auracast™ output from the media streamer and the ambient output from the TV's speakers. This arises from the serial actions of transcoding delay, delay in the streamer and decoding and video processing in the TV, which can add up to a total of several hundred milliseconds.

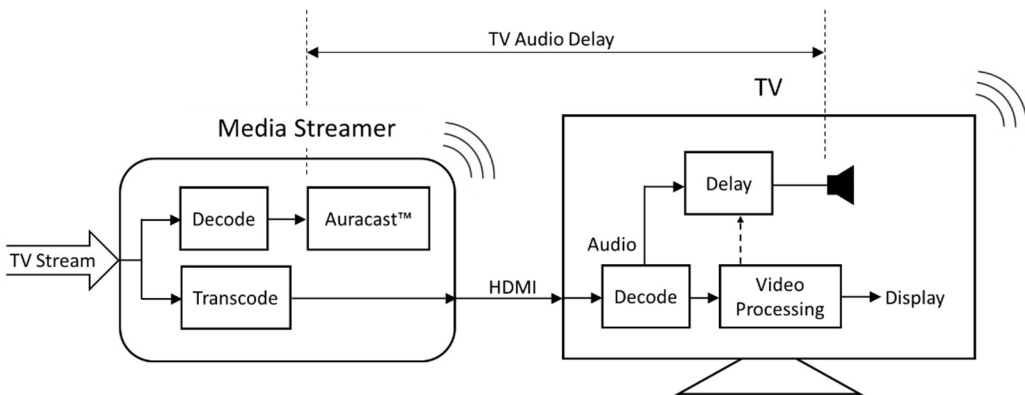


Figure 13.14 Audio delay caused by an external Auracast™ media stream with a TV

Media Streamer manufacturers can eliminate this by adjusting the value of Presentation Delay for their transmissions, setting it so that Broadcast Receivers render at the same time as the TV speakers. However, there is no feedback method over the HDMI connection to facilitate this. One option is to allow users to manually adjust a delay in the media streamer, although the Bluetooth LE Audio specifications do not provide a way to do this dynamically. An alternative is to include a microphone in the media streamer which can listen to the TV audio output during a configuration setup, correlate this with its internal, decoded audio stream and use this to set a Presentation Delay value that compensates for the difference. In most cases,

that should be a fixed value, although it might vary for different streaming services or TV formats.

The examples above only support a single language audio stream. As streaming services evolve to support multiple simultaneous languages in one stream, media streamers and TVs need to support a more complex architecture to allow the choice and output of multiple streams.

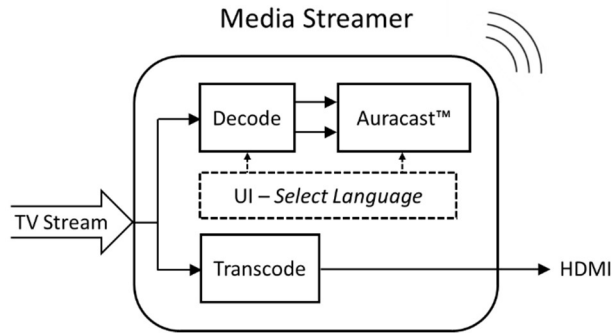


Figure 13.15 A media streamer supporting two different language audio streams

Figure 13.15 illustrates the architecture of a simple media streamer for two languages. It requires a User Interface option to allow the user to select which languages they want transmitted. This directs the decoder to output those two streams to the Auracast™ Transmitter, and also provides the transmitter with the metadata that it requires to label those streams. The same considerations apply to TV design for supporting multi-language audio streams.

13.4 The third device – The Broadcast Assistant

One of the biggest surprises to those of us who have been working on the Bluetooth LE Audio specifications has been the lack of companies making Broadcast Assistants, either as phone apps or stand-alone devices. That lack is hampering the availability of new broadcast applications, which were intended to be the hallmark feature of Bluetooth LE Audio.

In the last few chapters we've looked at how important the Broadcast Assistant is to the broadcast ecosystem, but it is worth reiterating its importance to the adoption of audio sharing, whether that's in a public venue, sharing music from your phone, or listening to the TV at home. The various Bluetooth LE Audio specifications provide all of the tools that developers need to innovate and make broadcast simple, but leaves the detailed implementation down to the developer. I've tried to include a variety of examples of how this can be done, from colocated Broadcast Assistants in TVs, QR Codes in phones and integrated Broadcast Assistants in battery boxes. The specifications are silent about these – they leave it up to developers' imaginations. Sadly, there has not been as much imagination as we had hoped.

In this section, we'll look at the power of collocating Broadcast Assistants with Broadcast Sources and also Broadcast Sinks. Although both may feel counterintuitive, they can have a profound effect on the user experience.

Section 13.4 - The third device – The Broadcast Assistant

13.4.1 Colocating Broadcast Assistants with Broadcast Transmitters

Making connections to home TVs presents a new challenge for both TV and headset developers. Many homes will have multiple TVs, all broadcasting Auracast™ streams, and family members will want to be able to connect to different ones as they move around the house. That is a very different paradigm to today.

One approach is to use QR codes, where a user can press a button on the TV or its remote control to overlay a QR code on the screen for a few seconds and scan it with their phone. This will instruct their earbuds to connect to the TV, as depicted in Figure 13.16. The disadvantage is that it assumes that the user is carrying their phone around with them.

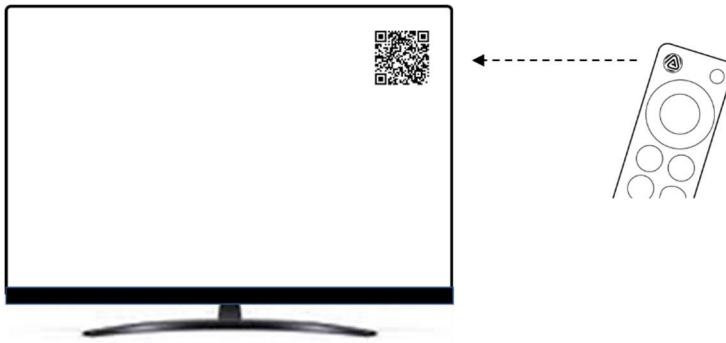


Figure 13.16 Using a remote control to overlay an Auracast™ QR code

A user could also use their phone as a Broadcast Assistant to scan for and connect to a TV but that seems unnecessarily complex for everyday home usage.

A less intrusive experience can be provided by collocating a Broadcast Assistant within the TV. Multiple family members can pair with the Broadcast Assistant, which can scan for Announcements from any bonded device within range. When a user enters the room and wants to connect, they would touch a button on their hearing aid or headset, which would announce that it needs a connection. The Broadcast Assistant on the TV would connect, write an Add or Modify Source operation to the headset, along with instructions to synchronise to the appropriate BISes. As home TVs are likely to employ encrypted streams, with session based Broadcast_Codes, the Broadcast Assistant in the TV should also write the current Broadcast_Code value, completing the information required for the earbuds to synchronize to and render the broadcast stream. Once the headset has successfully synchronized, it can terminate the ACL connection to the TV's Broadcast Assistant.

The use of a session based Broadcast_Code, keeps the audio content secure, so that others cannot eavesdrop on what a user is listening to. If friends want to share the TV, then overlaying the QR code is an effective way of allowing secure, session-based access. A TV should update its Broadcast_Code each time it is turned off.

The TV experience introduces a lot of new user concepts into the TV and Bluetooth

ecosystem, which will require innovation in User Interfaces. Developers need to concentrate on providing a simple and seamless way of evolving the current connection methods, to produce something which is intuitive and easy.

13.4.2 Colocating Broadcast Assistants with Broadcast Receivers

Many developers struggle with the concept of colocating a Broadcast Assistant with a Broadcast Receiver, as they see the Broadcast Assistant as a device which controls one or more Broadcast Receivers. However, there is an interesting use case, which is for sets of speakers which are expected to work in a broadcast environment without a dedicated device to act as a Commander. Although they may be a Coordinated Set, if they are able to act independently, they need to have their own method of synchronizing their behaviour. It's unlikely that they will contain a proprietary radio link in the way that earbuds do, so the simplest solution is to include a Broadcast Assistant into the speaker with the user interface.

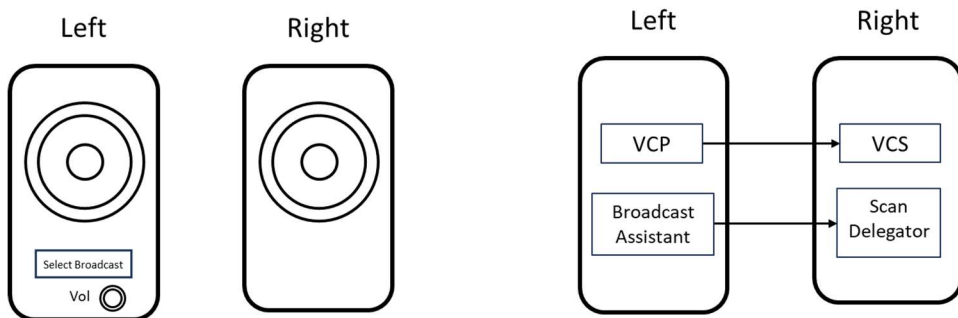


Figure 13.17 An example of a colocated Broadcast Assistant and Broadcast Sink in a pair of speakers

Figure 13.17 illustrates this situation. On the left is a pair of stereo speakers. The Left speaker includes a user interface which allows the selection of a broadcast stream, along with a volume control for both speakers. It also includes the Broadcast Assistant role to support the Scan Delegator functionality that is within the Right speaker. This implies that the two speakers are paired (normally performed at the point of manufacture) with an active ACL link between them when they are operating.

Once the Left speaker has selected a stream, it will write to the Broadcast Audio Scan Control Point characteristic on the Right speaker, instructing it to synchronise to the same Broadcast Source, using the BIS for the Right audio stream. No audio is sent between the two speakers – each synchronise individually to the same Broadcast Transmitter, rendering at the same time as directed by the Presentation Delay value in the BASE.

The Left speaker shown here includes a volume control which acts as a local control to its own gain, as well as controlling that of the Right speaker, using their VCP and VCS instances. The Left speaker will also contain a VCS instance (not shown in the diagram) to allow remote volume control. When it receives a remote volume change, it should not relay it to the Right speaker, as that should receive a command directly from the remote volume control. Both speakers will notify their current VCS values when there is any change.

Section 13.4 - The third device – The Broadcast Assistant

This configuration is useful for infrastructure applications, where multiple speakers can be controlled by a “master” speaker, providing an alternative to extending Coordination to multiple devices.

--- o0o ---

In the final chapter we’ll look at a few more application areas for Bluetooth LE Audio. The specifications really are so broad that almost anything you think of within audio can be done.

The Bluetooth LE Audio specification process is not complete. Multiple new enhancements are being worked on which will extend the flexibility of Bluetooth and the audio ecosystem.

Already, much more is possible than ever before, and much more that is currently being implemented. Nobody yet knows what the next audio killer application will be. As is so often the case, it may come from an unexpected direction. That is up to you.

Chapter 14. Bluetooth® LE Audio applications

This chapter serves both as a conclusion, as well as an introduction to readers who want to understand the possibilities offered by Bluetooth LE Audio before they jump into the detail. The whole point of developing Bluetooth LE Audio was threefold:

- To produce a lower power alternative to the existing Bluetooth Classic Audio profiles
- To catch up with proprietary extensions, particularly for True Wireless Stereo, and
- To support new audio applications

The last of those three was the big prize. The audio world needed a new wireless audio platform to allow innovation in audio. True Wireless Stereo (TWS) earbuds have shown the demand to move beyond basic speakers and headsets, while voice assistants have shown that voice has found a new lease of life. However, both were constrained by the limitations of Bluetooth classic audio. Something new was needed to make audio more universal and allow greater flexibility in how we can listen to it.

Designing from a clean sheet of paper allowed us to complement Bluetooth's existing audio use cases with some key new ones. Broadcast is the child of the telecoil. The telecoil system of inductive audio loops for hearing aids has been around for almost seventy years. It performs well, but it is remarkably basic. It is mono, has a very limited audio bandwidth, and you can only hear it if you're located within the confines of the inductive loop. Most people were unaware of the experience, even if they had a hearing aid. Whilst the aim of Bluetooth technology was to provide a more capable successor, it very quickly became obvious we could expand significantly on the user experience of telecoil and bring that to a new audience. It was no longer just for the hearing impaired, but a technology that allowed sound to be part of an infrastructure, as well as supporting ad-hoc sharing wherever you are. It has become the hallmark feature of the new Bluetooth LE Audio specifications, recognised in the Bluetooth SIG's Auracast™ brand, which promotes an interoperable broadcast experience for everyone.

The concept of universal, interoperable broadcast and audio sharing is still sufficiently new that many in the industry find the concept challenging. In this chapter we'll look at some of the use cases it enables and how users are likely to use it. It's probably fair to say that in a few years, Auracast™ will be seen as the biggest innovation in audio technology since the introduction of stereo back in the 1950s.

14.1 Changing the way we acquire and consume audio

Changing the ways we acquire and consume audio are important points for developers to understand. For most of the last two decades, the evolution of personal audio has been driven by mobile phones. Initially, we stored files that we'd acquired from music sharing networks on our phones. More recently, with the growth of audio streaming services, the phone became

the router supplying music to our ears on demand. However, that experience largely depended on an interaction with the phone to select what we wanted to hear. With the features of Bluetooth LE Audio, and the predicted availability of multiple Bluetooth LE Audio sources, that is going to change. Those sources may be personal, in the case of our phones, laptops and TVs; public, such as a Broadcast Transmitter that's installed in a restaurant, pub, gym or office; or commercial, as in a cinema or when we're listening to a live performance.

An issue with replicating the telecoil experience is that a Bluetooth transmission is not confined by its installation area. Being wireless, it penetrates walls, meaning that people in adjoining rooms and spaces can also access any broadcast audio. In many situations, such as public halls and places of worship, that's not a significant problem, because the only source of broadcast audio will be the one that is relevant for that particular venue. However, in other situations, such as TVs in hotel rooms, or within conference centres where there are multiple meeting rooms, that becomes a major issue, as broadcasts will overlap, with the result that people will have difficulty in understanding which broadcast is the one they want to connect to, and potentially hearing things which they shouldn't.

That led to the implementation of encryption within a broadcast stream, so that only a user with the correct Broadcast_Code to decode that stream is able to listen to it. Although the broadcast streams overlap, the Broadcast_Code provides an access mechanism where only authorised listeners can decode a particular broadcast Audio Stream. That's akin to how Wi-Fi works today, where users are given an SSID name to identify a particular Wi-Fi network, and then need to key in an access code to be allowed to connect to it. Whilst people have become used to doing that with Wi-Fi, it's a pretty basic and often frustrating user experience. Some early Auracast™ implementations copy that usage model, but everyone wanted Bluetooth LE Audio to be a bit smarter.

Tackling that needed a better mechanism for a user to access the code than a scrap of paper on a coffee shop table or a notice on the wall. It led to the development of the Broadcast Assistant, providing ways for that information to be acquired from a Broadcast Source. As the specifications developed, it quickly became obvious that this provided a very powerful mechanism for users, both to pick up the Broadcast_Code, and gain access to individual broadcasts in a far more flexible manner than we have ever seen with previous Bluetooth connections.

Broadcast Assistants allow the display of much more than a list of Broadcast Transmitter names. They can display the content of what is being broadcast, as well as its language and audio quality. The amount of information in the advertising streams that the Broadcast Transmitter produces allow users to make decisions on automatic connection to their favourite audio sources, which can bring in other contexts, such as location. Although everyone expects that the first Broadcast Assistants are likely to be implemented as phone apps, there are many other options, as we saw in Chapter 12. As these alternatives come to market, users will find they are able to do far more in terms of selection and control without touching their phones.

Section 14.2 - Broadcast for all

It will be interesting to see what effect that has on the devices we carry with us and wear. If we can do more without touching our phones, their importance as the central device for much of our communication may wane. In turn, as voice becomes increasingly natural as a means of command, new applications may develop which use audio as their primary interface, with no need to interact with a screen. Smartphones were never going to be the final step in the evolution of our personal communications – some other product will emerge, as smartphones did themselves. The new capabilities that Bluetooth LE Audio brings to the equation, allowing greater ubiquity for voice and audio, may hasten that change. Some of the first of these “phone replacement” products, riding on the AI bandwagon, have already appeared. They still have a long way to go, but voice is beginning to challenge the current ubiquity of touch.

The ability to use multiple Broadcast Assistants means that wearable devices gain increased utility. Whilst not all of that will relate to audio, it increases the reasons for purchasing and wearing them. That will help the wearables industry as a whole, which is still struggling to reach the volume or customer interest that it had hoped for.

All of these changes will take time. Some will happen earlier, others later. Some may come in implementations which fail to convince users, but then get successfully reinvented a few years later. That’s been the case for most new personal technologies. What is clear, is that they will change the dynamics of the audio ecosystem, giving far more opportunity for audio manufacturers to innovate and chip away at the de facto status of smartphones. In this chapter, we’ll explore some of the different scenarios for Bluetooth LE Audio, look at the opportunities for new ways of presenting audio sharing services, determine what is needed to enable them, and assess what this may mean, both for phone design and for the design of other products.

The Bluetooth LE Audio specifications will not, by themselves, transform the way we use audio. To be effective, many different people working in the industry – hardware developers, apps developers, service providers and UX designers need to understand the possibilities and take on board how to make these new audio experiences compelling, letting users discover how audio can fit in with their lives in new ways, and then build on that to deliver new ways of using audio. An important point to realise is that we can now use audio to control audio. A whole new set of user experience possibilities are emerging, which need designers to move beyond the fixed peer-to-peer connectivity that they’ve been living with for the last twenty years. The Bluetooth LE Audio specifications provide a very powerful toolkit to build new experiences, but those experiences need to be invented and allowed to evolve.

14.2 Broadcast for all

There is no question that both users and venues who currently use telecoil will welcome the advent of Bluetooth LE Audio as the next step in hearing reinforcement. The quality is considerably higher, and installation costs should be significantly lower. For hearing aid users, it promises a much wider range of places where they can connect for hearing reinforcement.

For building owners and architects, it should mean that hearing reinforcement becomes a standard feature of new buildings. (It's a sad fact that the lack of telecoil in many new buildings has more to do with a lack of understanding from architects than the cost of the current technology.)

This should all happen naturally, as broadcast products become available, helped by an ongoing promotion of Auracast™ by the Bluetooth SIG and the hearing aid industry. However, users will need new hearing aids that contain Bluetooth LE Audio to pick up these broadcasts. Currently only a small percentage of hearing aids contain any form of non-proprietary Bluetooth wireless technology, and it will take time for a critical mass of users to emerge. Compared to consumer volumes, the hearing aid market is relatively small – it will probably take at least five years to attain ten million users of new Bluetooth LE Audio hearing aids, not least because hearing aids need medical approval before they can be sold, which slows down time to market⁸¹. This raises the interesting question of whether consumer products will drive the initial roll-out of audio broadcast infrastructure, which will benefit everyone, regardless of whether they have hearing loss.

14.2.1 Democratising sound reinforcement

Today, providers of public broadcast/telecoil services are thinking predominantly about people wearing hearing aids when they design their products. These services most commonly replicate and reinforce an audio announcement with low latency. Very few people are thinking about whether this will be relevant to people without hearing loss, and how they can be extended. The likelihood is that many users of earbuds and headphones would find them beneficial. If that is going to happen, then most people, certainly in the short term, are likely to find and select them using apps on their smartphones. So, the first step will be for the phone operating systems to expose the information about these broadcasts, allowing users to select and listen to them. In other words, we need to see app interfaces which look something like those in Figure 14.1, mimicking what is already done today for discovering and pairing to Wi-Fi and Bluetooth devices.

⁸¹ As this book was going to print, Apple announced that they had achieved FDA approval for one of their AirPods devices for use as a hearing aid for people with low and moderate hearing loss. This may help persuade many more people that they would benefit from wearing hearing aids.

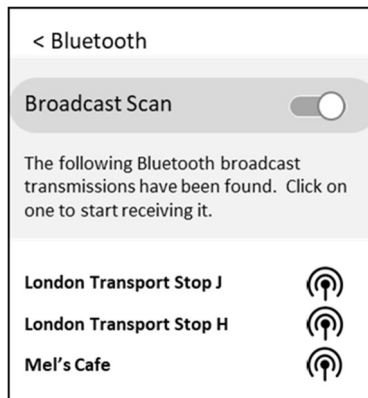


Figure 14.1 Simple user interface to find Bluetooth LE Audio Broadcasts

Although this is a format that most users will be familiar with, it's still more difficult than it needs to be. The example of Figure 14.1 illustrates the fact that one of the first applications is likely to be in public transport. Many bus stops have announcement boards, but unlike train platforms, don't make public audio announcements because they are intrusive on a street. National guidelines in many countries already require telecoil to be incorporated in new public infrastructure, and are likely to include Bluetooth LE Audio in their future recommendations. As these appear, good transport apps should start to include support for them, so that travel apps automatically detect the presence of an appropriate audio stream and ask the user if they wish to listen to it, saving them the inconvenience of going into their Bluetooth settings to search for it.

With this information, applications which are scanning on behalf of your earbuds or hearing aids can obtain a lot of information to direct them if they want to start embedding broadcast audio into their user experience. Equally, the providers of broadcast audio information need to think carefully about how to use it and how best to support applications. The primary use of most public broadcast will remain sound reinforcement, where the broadcast audio is designed to be low latency, so that it can be received and rendered alongside ambient audio. It's now possible to include multiple languages for those announcements, but they should be scheduled so they don't conflict with an ambient announcement in another language, otherwise they will be more difficult to understand. It's a small, but obvious nuance, but one of many that developers will need to learn.

Broadcast stream providers also need to think about the information they use to tag the streams. As you move from a bus stop onto a bus, your application should track the loss of the bus stop as a Broadcast Source and switch to the Broadcast Source on the bus you're travelling on. In somewhere like London, with a high density of buses, it's quite possible that you may be next to another bus on the same route, but travelling in the opposite direction, so an application should contain enough basic intelligence to detect that it has connected to the right transmitter. As long as the Local Device names and ProgramInfo values are sensibly ascribed, it should be straightforward to determine that. However, this needs service

providers, application developers and equipment manufacturers to work together to give their users the best experience.

Most public broadcast streams will be mono voice streams, which can be adequately encoded using the 16_2_2 QoS settings, using very little airtime. The messages are likely to be infrequent – in the example of a bus, either the announcement of the arrival of a bus, or, when you’re on it, an announcement of the next stop. If your phone has sufficient resources, a travel app should be able to monitor the appropriate BIS, mixing it in to any other Audio Stream at the point where it detects the BIS contains an audio message and ignoring it when there are only null packets.

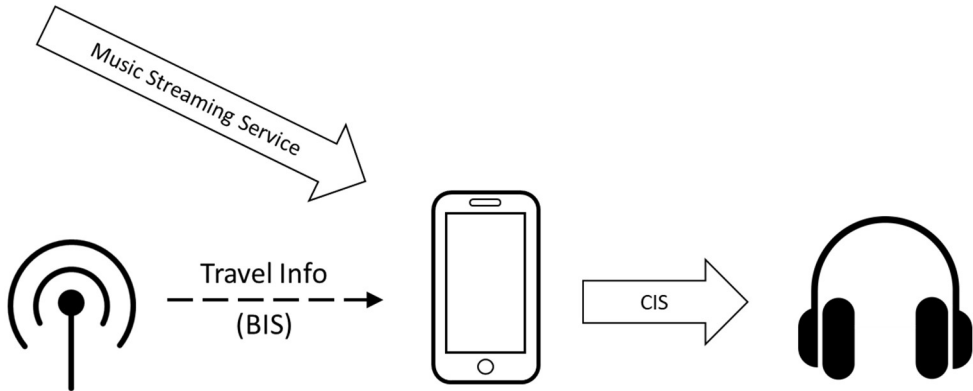


Figure 14.2 Mixing broadcast announcements into an audio stream

Figure 14.2 shows the basic setup, where a phone is running both a music streaming app and a travel app, which is monitoring the local Broadcast Transmitters for relevant audio announcements. Figure 14.3 shows how the phone would combine audio from the different sources into a single stream which is encoded and sent in a single CIS. It makes the point that an Initiator is free to mix whatever audio channels it wants to place in a stream to send to the earbuds.

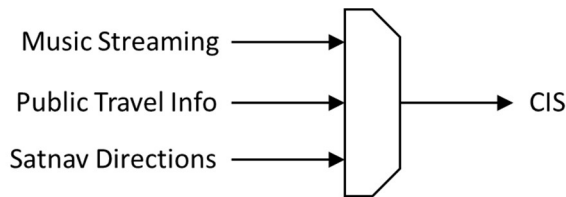


Figure 14.3 Mixing application specific audio streams into a CIS

Given that people are starting to leave their earbuds in for longer, using the transparency mode to hold conversations, this is a useful way of receiving relevant announcements, particularly if they are “silent” foreign language announcements.

Section 14.2 - Broadcast for all

14.2.2 Audio augmented reality – the “whisper in your ear”

The example above of mixing broadcast travel announcements into other information generated by a travel app is the entry point into using broadcast audio as an element of augmented reality. The initial promises of augmented reality and virtual reality have proven to be rather disappointing, with few successful applications outside specialised business ones and products for stay-at-home gamers. Audio may offer a more acceptable entry step, particularly for everyday augmented reality.

The advantage audio brings is that it is far less obtrusive. It's the little whisper in the ear that helps you do whatever else you're doing, with no need to purchase or wear what are often inconvenient wearable tech products. We are already seeing multi-axis sensors incorporated into earbuds which can detect your head position, and hence know where you're looking. Combined with information from Broadcast Transmitters and spatially aware applications, these allow some innovative new audio applications where sound can start to merge into your everyday experience, without the need for any special AR or VR hardware. Audio directions can tell you which way to turn and where to look. It seems likely that satnavs are going to become much more personal. A lot of the underlying technology for these already exists. It is just waiting for the potential of Bluetooth LE Audio to make it possible.

14.2.3 Bringing silence back to coffee shops.

The examples above show how existing telecoil applications can be expanded to a wider audience and integrated into today's smartphone apps. There is likely to be a parallel evolution of broadcast infrastructure in new areas. One that is getting a lot of attention is employing broadcast to provide background music in cafes and restaurants.

At some point in the past, somebody thought it was a good idea to equip most cafes, bars and restaurants with background music. The design chic of the past decade has largely been to remove any furnishings that act as noise absorbers, making conversation increasingly difficult and raising the background noise level. That generates a positive feedback loop where customers start talking more loudly, so the staff turn the music up, resulting in customers having to raise their voices and turning what should be an enjoyable experience into a far less pleasant one. Restaurant reviews now routinely include a measure of noise to show whether it's possible to hold a conversation. Despite negative consumer feedback, the music remains. It would be nice if Bluetooth LE Audio could effect a change.

Parts of the hospitality industry believe that there is an obvious place for Bluetooth LE Audio to replace loudspeakers, providing music for those who want to listen and reducing the noise levels for those who want to talk.

The addition of hardware can be extremely simple. All it needs is a Bluetooth LE Audio broadcasting module fitted to the output of an existing audio system. Chip vendors are already developing reference designs for units like this, and devices like are becoming available for a relatively low price. Product designers should make sure that they support the Auracast™

Simple Transmitter Best Practices Guide and can be easily configured by the venue owners, so that they are easy for customers to discover.

14.3 TVs and broadcast

From an early point in the Bluetooth LE Audio development, the TV industry became very excited about the possibility of connecting TVs to earbuds, headphones and hearing aids. Although TVs can use unicast, there is an obvious limitation to its scalability, as it requires separate CISes for every listener. Hence, the consensus is that most TV usage will use broadcast, adding encryption to ensure that only authorised users can decode the audio content. In this section, we'll look at the requirements for the three most common application areas.

14.3.1 Public TV

Today, a large number of publicly installed TVs are silent. It's not because they have no audio output, but because they are installed in areas where ambient audio would be annoying or conflict with other audio. Common examples are:

- Airports, where operators don't want users to be distracted from hearing flight and security announcements,
- Gyms, where multiple TVs display different channels, but are silent, as multiple conflicting audio from them would be cacophonous,
- Pubs and bars, where the sound from even one TV (and there are often multiple TVs with different channels), would disrupt normal conversation, and
- Outdoor TV installations, where the sound would need to be intrusively loud to be heard above the background noise.

These installations are obvious candidates for broadcast audio. They don't need encryption; they just need a Broadcast Transmitter. That could be a simple plug-in device attached to their TV's audio outputs, or a built-in Bluetooth LE Audio Broadcast Transmitter. It needs to allow a device identity to be entered so that a user can match an audio stream with a TV. That could be a name or a description of the content that makes it obvious to the potential listener that the two are related.

As we've seen above, many of these devices, including those integrated into TVs, will probably be managed, not least to allow specific messages to be mixed into the audio stream. For example, in an airport, flight announcements would probably be mixed into the TV's audio stream, so that someone listening to a TV would never miss them. This is where we will probably see the start of multiple language streams. If multiple language soundtracks are available for a TV broadcast, they can be sent in separate BISes on one BIG. Using a 24kHz

Section 14.3 - TVs and broadcast

mono coding, it should be possible to provide six different mono language streams from one Broadcast Transmitter. Each input stream would mix the flight announcements with the audio input for that TV program. These would normally be mixed to ensure they don't conflict with any ambient announcements which might degrade their intelligibility, as that could distract the listener, but that's just good management of audio announcement. A user can set their scanning application to select specific languages, so that these are presented preferentially. Alternatively, their Broadcast Assistant could show all of the language variants which are available and let the user choose the one they want to hear. This requires the phone's APIs to expose this information and application developers need to understand how to use it.

14.3.2 Personal TVs – at home

Personal TV has different priorities. Many TVs already support Bluetooth Classic Audio profiles, but that is generally limited to connecting to a soundbar, a single set of earbuds or a headphone. The reality is that in most homes, multiple members of the family or friends will be listening. Although TV manufacturers could do a like-for-like transition by moving the Bluetooth LE Audio unicast, that will quickly hit an airtime limit of a few users. It makes far more sense to move to broadcast.

Unlike the public TV cases described above, most users would not want their neighbours to hear what they are listening to, so for personal TVs (and other audio sources in the home), the expectation is that broadcast Audio Streams would always be encrypted. That means that there needs to be a simple mechanism for the user to obtain the Broadcast_Code to decrypt the Audio Streams.

This is where the Broadcast Assistant comes in. Personal audio sources, which expect to be used regularly by multiple users, would require each user to pair and bond with them, as they currently do with Bluetooth classic audio. That provides a long-term trusted connection. When a user comes within range of the TV, they would ask their earbuds to connect and the Broadcast Assistant in the TV would inform them of where to find the broadcast stream using PAST, followed by the secure transfer of the current Broadcast_Code. This is an important point - once a user has paired, they no longer need to use their phone to connect to an encrypted broadcast. When the user comes within range of the TV, the basic LE connection can be automatic. A pair of earbuds can be informed of that connection with an internally generated audio message, alerting the user to the presence of audio source. If they want to connect, they press a button or perform whatever gesture the earbuds require, allowing them to start audio streaming. For users walking between rooms with different audio sources, it's an elegant way for them to connect to the nearest source. Notifications are provided to all Broadcast Assistants informing them that the earbuds are dropping synchronization or synchronizing to a new source. It means that every connected device which is involved keeps track of the current status, making it possible to engineer a very elegant solution.

To enhance security, personal TVs should regularly change their Broadcast_Code, typically whenever the TV is turned on. Bonded devices will automatically receive the new code when they reconnect.

When friends come around and want to connect with their earbuds or hearing aids, the TV owner would put their TV into a Bluetooth Audio Sharing mode, (which would ideally be a button on their TV remote control, not a menu item buried many layers down on the TV). That would activate the advertising train associated with the broadcast stream, which would provide a method for the friend to obtain the Broadcast_Code. It is expected that most manufacturers will support QR Codes that implement the Bluetooth Audio URI specification.

14.3.3 Hotel TVs

Anyone who has ever been disturbed during their stay in a hotel by the TV in the neighbouring room being far too loud will welcome the advent of Bluetooth LE Audio in TVs, as it allows the occupants to use them silently. It's an ideal application, but carries the same problem of Bluetooth transmissions penetrating through walls, floors and ceilings, exposing what you are watching to all of your neighbours.

The same approach can be used as in personal TVs, where a user is given a Broadcast_Code which is valid for the duration of their stay. The hotel could provide them with a passcode to enter into their Broadcast Assistant, or a QR code to scan. However, in managed installations like the hotel there's an even simpler way to provide access, which is to incorporate the Broadcast_Code into the hotel phone app, so that as soon as the customer checks in, their phone would be provided with the correct information to access the TV in their room.

14.4 Managed Audio Services

The scenario above, where the broadcast credentials are integrated into a user app is a major differentiator for managed audio/visual services. Whether it's access for a hotel TV, the soundtrack at a cinema, audio access at a theatre, or multi-language streams for a conference room, all of the necessary credentials can be set up in advance on a user's account, allowing instant access from a smartphone app. This opens up completely new revenue opportunities for audio infrastructure suppliers.

14.5 Phones and broadcast

In the early days of Bluetooth LE Audio development, broadcast was mainly seen as an infrastructure application, broadcasting to large numbers of people. As the potential of the technology became better understood, there was growing excitement about what it offered for smartphones. What excited people most was the ability to share music on their phone with their friends.

The use case is a simple one. It has been around since we first started storing music in personal devices. Back in the days of Sony's Walkman, you'd see pairs of people sharing their earbuds

Section 14.5 - Phones and broadcast

to listen together to the music. As we've progressed from portable music players to streaming music on our smartphones, the technology for sharing has not advanced. In contrast, it's become more difficult with the removal of the 3.5mm audio jack from smartphones, as the simple sharing of wired earbuds is no longer possible with many handsets.

Bluetooth LE Audio broadcast solves that problem. If you're listening to your favourite music and your friends come along, you can ask them to join you, by transitioning from a personal stream to a broadcast stream. That sets your phone up as a Broadcast Source which your friends can find. Unless you want to be a public Broadcast Transmitter, your application will almost certainly want to encrypt your audio, so your phone application needs to distribute the Broadcast_Code, which can be easily accomplished with a QR code. Equally, any of your friends can take the role of Broadcast Transmitter to share their favourite music. It's a process which can continue for as long as any of them want.

In some cases, the stream doesn't need to be encrypted. Anyone wanting to set up an ad-hoc silent disco just needs to start broadcasting and let people know that the stream is available.

14.5.1 Party Speakers

During the development of the Bluetooth LE Audio specifications, some of the engineers became very excited about the prospect of using it for party speakers⁸². Broadcast Transmitters have no knowledge of how many Broadcast Receivers are connected – they just need to be within range. The range for Bluetooth LE Audio is much greater than people associate with Bluetooth, typically covering 50m or more. This means that friends can bring their own speakers to a party and set them all to render together, whether that in a public venue, a garden or a beach.

Any Bluetooth LE Audio speaker can be set up by its owner using a Broadcast Assistant. Some companies are also designing party speakers with a user interface that can scan for available broadcasts, providing an integrated solution.

14.5.2 Whole Home Sound Systems

In the same way that multiple speakers can be used to render a broadcast, the Bluetooth LE Audio technology is driving the next generation of in-home audio systems. In many cases, the range of the master Broadcast Transmitter will be sufficient to cover the whole house. Where it is not, relay transmitters can be designed with minimum latency. This is possible because the encoded LC3 packets can be retransmitted without any need for sampling or decoding, resulting in relay latencies of little more than 10ms.

As each speaker has its own volume control, this allows sophisticated volume control across a house, which can include presence detection in each room.

⁸² It was unkindly suggested that they hoped this might get them invitations to more parties.

14.6 Microphones

Although Bluetooth headsets and earbuds have always included microphones, there have been very few dedicated Bluetooth microphones. The reason for that is not difficult to discern – the latency of A2DP is too great, and the audio quality of the lower latency HFP is not good enough. Bluetooth LE Audio has solved both of those problems, opening up the market for a range of interesting microphone applications.

14.6.1 Phones as microphones

Phones can take advantage of Bluetooth LE Audio without being phones. Many people with hearing loss like to use table microphones to help pick up voices in a meeting or around the dinner table. Once your phone has Bluetooth LE Audio, an application can set it up as a private Broadcast Transmitter, broadcasting its microphone pickup to everyone else around the table. It uses no cellular functionality, but acts as a local community microphone. It's a feature which Apple has implemented in their proprietary "Listen Now" service, but it now available with enhanced audio quality and complete interoperability.

14.6.2 Portable PA systems

The cost of adding Bluetooth LE Audio into a speaker or microphone is considerably less than the cost of a cable for a traditional speaker or microphone. As we saw in Section 12.1.2, such a microphone with a couple of speakers provides everything needed for a PA system. It is totally portable and free of any cables. Whether it's a church, a karaoke venue or a public meeting, it provides a complete audio system which can be set up in seconds to support everyone, regardless of whether or not they have hearing loss.

14.6.3 Personal silent discos and yoga

It is always interesting to hear consumer feedback as a specification is being developed, as it can often be unexpected, suggesting different use cases. During the course of writing and talking about Bluetooth LE Audio, two of the applications I've constantly been asked about are for silent discos and silent yoga. There is a surprisingly large industry providing proprietary microphones and headsets for all sorts of group applications, where there is a leader or instructor providing instructions to a group of people. The disadvantage of these is that the person running the group needs to purchase enough headsets for everyone to use, issue them to participants, then retrieve them at the end of the session. That's a significant cost, with the risk that some will go missing at each session. To keep costs down, current devices cut corners on performance, giving limited range and poor audio quality. Bluetooth LE Audio means that participants can use their own headphones or earbuds, making silent group activities more convenient, easier to set up and better audio quality.

As with current "instructor" microphones, an instructor's microphone can allow other music streams to be mixed with their voice.

14.7 Personal communication

Whilst all of the broadcast applications described above can be accessed by individuals, most are likely to be used by groups of people, and the interface designs for them should be designed with that in mind. However, there are applications which will be predominantly designed for individual conversations. Once again, these mimic many of the telecoil applications, where a small inductive loop is used to provide an audio feed for a single person. These are typically found at ticket desks, taxis, banks, supermarket checkouts and hotel receptions. These use a static microphone to pick up the hearing aid wearer's voice, and a telecoil loop to return an audio stream from the other person to their hearing aid. What is important in this application is that the broadcast audio stream is private and never mixed up with another one nearby. While the conversation can always be heard by someone standing nearby, you do not want it to be picked up by someone several metres away. Nor do you want the wrong stream to be picked up. So, once again, encryption and authentication are necessary.

14.7.1 Scan 2 Hear – making it simple

For people with hearing loss, small telecoil loops have been developed for use in personal applications, such as reception desks and ticket booths. These allow a hearing aid wearer to stand in a specific location and hear the voice of the person they are talking to. These telecoil devices usually include a directional microphone to capture the voice of the customer, which is reproduced in the receptionist or ticket seller's headset.

With Bluetooth LE Audio, it is recommended that the audio stream is encrypted to prevent it eavesdropping. This means that the hearing aid wearer needs to acquire the Broadcast_Code, which should be updated for each new connection. This can be achieved very simply by having a local display on the reception desk which generates a new QR code. Each new customer can scan it, after which their hearing aids will be able to decode the conversation. It makes sense to limit the output power of the Bluetooth LE Audio transmitter, but the range still allows multiple family members to connect and participate in the conversation.

In the future, the same underlying Bluetooth Broadcast Audio URI specification is expected to be used to allow a similar experience using Near Field Communication (NFC), where they just need to tap their phone on a touchpad to be securely connected.

Although nothing has yet been specified, industry interest is focused on the use of NFC to provide a simple "Tap 2 Hear" interface, with the user tapping their phone, or any Commander device, onto a touchpad. That contact will transfer the information that the hearing aids or earbuds need to find the correct Broadcast Source and the appropriate Broadcast_Code. As soon as that is done, the conversation can begin, with the Broadcast_Code and BIG details remaining static for the duration of the session. It provides an attractive and simple user experience, and is applicable for all types of personal connection, whether at a supermarket checkout, accessing the right conference meeting room, or even connecting to the broadcast stream in a theatre or cinema. It is equally applicable to Audio Sharing, where friends would

just need to tap your phone to share music, or to gain access to a Private TV.

14.7.2 Wearables take control

Although the Bluetooth Classic Audio profiles allow for remote control on separate devices, there has been very little use of these capabilities. These have mainly been limited to carkit functionality, with an occasional foray into wearable devices. With Bluetooth LE Audio, that is likely to change. There are a number of reasons for that.

One of the main drivers for remote controls is the continuing growth of earbuds. As the hearing aid industry discovered several decades ago, adding user controls to something as small as an earbud or hearing aid is not easy, either for the manufacturers, or the customer. As a result, customers have been encouraged to use phone apps to control their audio. However, constantly taking your phone out and opening the appropriate app is far from being the best user experience. If something is too loud, you want to reduce the volume as quickly as possible. That plays to adding earbud volume controls to more accessible devices and wearables are an obvious candidate. They provide far faster access than a phone app, and adding this functionality is a useful differentiating feature.

The control features within the Bluetooth LE Audio specifications make it much easier to incorporate remote controls into other devices and to have as many of them as a user wants. These are low power products which can easily run off coin cells, so can be low cost, and will be interoperable, allowing any Bluetooth product to integrate the features. As a result, we expect to see a trend for wearable devices to implement this functionality, whether that's wristband, smart watch, glasses or clothing. It may be a feature which increases the usefulness of wearables for many users, bringing life back to what has become a relatively moribund product sector. It may also result in a comeback for MP3 players, allowing people to take their music with them when running or jogging, and leaving their phone at home.

14.7.3 Battery boxes become more important than phones

It's worth adding a few words on the humble battery charger box for earbuds. It's a necessary accessory which was developed alongside earbuds to give them the semblance of a useable battery life. The first generation of earbuds mostly struggled to run for an hour before they needed recharging. The battery box was a neat idea, both to hold them safely, but also to recharge them constantly, giving users the perception that they would run for a day. Since those early products, the battery life of earbuds has improved significantly, with models claiming up to 10 hours of continuous music (albeit with processing features like Automatic Noise Cancelling turned off). Bluetooth LE Audio will increase the basic battery life, although manufacturers will probably take the opportunity to add more features which suck up power. Regardless of that, battery boxes are here to stay, not least because they provide a very important function alongside their charging capability, which is a container to stop you losing your earbuds.

Section 14.8 - The future of earbuds

Many battery boxes already contain a Bluetooth chip to help with pairing, and to provide an audio stream in situations where one doesn't exist, such as on an aircraft. Here, the battery box can plug into the 3.5mm jack provided by the aircraft's personal entertainment system, transmitting sound to your earbuds. Bluetooth LE Audio's lower power and lower latency make it the ideal choice to replace Bluetooth Classic Audio in these applications. However, the battery box is also ideal for many remote control functions. Unlike earbuds and hearing aids, it is big enough for buttons, so is ideal as an easily accessible volume controller. It can also act as a Broadcast Assistant, scanning for available Broadcasts. In general, it will provide a much faster and easier interface than getting your phone out and opening an app, because the Bluetooth LE Audio control functionality is always there as buttons.

Almost all manufacturers already supply their earbuds pre-paired with the battery box. When you incorporate a Broadcast Assistant into the battery box it provides an amazing user experience. The first time they use the earbuds, they just pop them in their ears, select a broadcast on their battery box and are instantly listening to music. No pairing needed. It's the way that wireless audio was always meant to work.

Designers are so used to everything revolving around smartphones, they can forget about how else they can create memorable experiences. Because of its relative simplicity, the humble battery box is often overlooked. However, its accessibility and small size, fitting into pockets that won't hold a phone, make it an attractive alternative for consumers.

14.8 The future of earbuds

As we saw in Chapter 1, earbuds have become the fastest growing consumer product ever. That gives manufacturers a challenge to build on their popularity and offer new experiences for consumers. Despite their popularity, developers are struggling to be particularly innovative, largely because of the constraints imposed by A2DP and HFP. Bluetooth LE Audio offers new ways to enrich consumer audio applications and enables new solutions which will add momentum to the market.

14.8.1 Audio augmented reality

Consumers have been offered the promise of virtual and augmented reality for over 30 years⁸³, but it has been slow in arriving. Many companies continue to pursue virtual reality headsets as the next big thing in consumer electronics, but the mass market has eluded them. Apple's Vision Pro is an amazing example of what can be achieved, but it is difficult to consider it as a mobile device which will be used outside a building.

Augmented reality – VR's little brother, has been more successful. Rather than full headsets, these are typically glasses that overlay video information or provide additional audio feedback on what you are looking at. Many of the applications have been in industry, from surgical

⁸³ Sega's VR headset was demonstrated at the Consumer Electronics Show in 1993,

training to equipment maintenance, but these are now beginning to appear as affordable consumer products.

Smart Glasses developers are beginning to understand the value of audio as the feedback mechanism. Rather than using expensive (and often distracting) technology to superimpose video on what you're looking at, they use audio to add the feedback. It's a subtle whisper in your ear which may be telling you which way to turn, the history of what you're looking at, or translating a sign. It currently relies on Meta's AI in the cloud, but is an obvious candidate to incorporate information from Auracast™ Transmitters, whether that's on public transport or in public venues.

A lot of the development for this audio feedback is being driven by the needs of partially sighted, who understand how powerful audio feedback is. In the same way that Bluetooth LE Audio was driven by the hearing aid community, it's a recurring example of how designing for accessibility often ends up benefiting everyone.

14.8.2 The growth of biometrics

Back in Chapter 1 we looked at some of the early aspirations for earbuds, which went far beyond audio. It's well known that the ear is an ideal location for biometric sensing. Unlike the wrist, which constantly moves around, the ear is relatively static. It's also close to blood flow and provides a good site to measure core temperature. While the early pioneers of hearables were keen to include multiple sensors, the applications and consumer enthusiasm weren't there – users just wanted to listen to music.

There is still a long way to go to turn the ear into our personal health monitor, but developers are beginning to reconsider how biometric data can enhance and differentiate the listening experience. The first foray into this is measuring heart rate, predominantly for runners. As well as a performance indicator, it can be used to select music or speed up an existing track to enhance an exercise regime. As sensors become smaller and more effective, other applications will follow. These are likely to concentrate on how the listening experience can be modified, rather than being purely capturing biometric data. That's something which has had relatively little attention. Audio manufacturers have largely limited themselves to volume, noise reduction and graphic equalisation. The advent of podcasts has helped popularise playback speed adjustments at constant pitch, so that a programme can be timed to match a commute journey, but we have yet to see much more than that. There is plenty of scope for innovation, using the biometric data from our ears to manipulate what we hear. It's not yet clear from which direction these developments will come, but the pace of evolution of hearables will certainly speed up.

14.8.3 Professional hearing aids

As a final example, I've noticed an interesting trend since the first few Bluetooth LE Audio hearing aids have come to market. Employees of the companies making them have started wearing them. These are people who haven't worn hearing aids in the past and have not necessarily used earbuds. But they've taken to Bluetooth LE Audio hearing aids because

Section 14.9 - Market development and notes for developers

they're lightweight, have great battery life and are convenient for calls. That's three things which are still a major struggle for consumer earbuds. The wearers report that they have no issue wearing them at work, because they're largely invisible; were they to wear earbuds, it would look as if they're ignoring their colleagues – constantly having to take them out to have a conversation. Surprisingly, they also say that they listen to more music, because it's so easy.

Once again, it feels that the hearing aid industry has something to offer everyone, although at the moment it's not obvious that they realise this. Once they do, we may see a new market segment for professional earbuds based on the performance advantage of hearing aids. Hearing is a spectrum and we should be taking the best of technology to support all users, whatever their level of hearing loss. It feels that we are at the point, where some of the superior technology of the hearing aid industry should be adopted in consumer devices.

14.9 Market development and notes for developers

There's a simplistic view that broadcast infrastructure in buildings, theatres, hotels and cafes will happen because the cost of a Bluetooth LE Audio broadcast devices will be cheap, so people will buy them from eBay and Amazon, plug them in to their existing audio system and put up a Bluetooth Auracast™ sign. That will certainly happen. It's what happened in the early days of Wi-Fi, when venue owners did exactly that. With Bluetooth LE Audio, it should be easier, as you don't need to install an internet connection, you just need a cable to attach it to your existing audio system.

However, with Wi-Fi, many venue owners discovered that it was easier to work with a service provider, who could install it, manage it and provide support for the venue and the customers. The same model is likely to appear with Bluetooth LE Audio. I suspect that we will see Wi-Fi Access Point providers selling access points which include Bluetooth LE Audio, so that a single device can manage both. There are similar opportunities for the companies currently making and installing telecoil loops, where they will see their business open up to a far wider range of customers. It will need apps developers to be aware of how broadcast works to provide the user interfaces.

On which front, it is important to point out that these applications will only happen when silicon, operating systems and application developers understand the full potential of Bluetooth LE Audio and include support for the features which enable these different use cases. In the early days, some of these may not be possible, as developers naturally concentrate on releasing what they consider to be the most obvious applications. Over time, as the market learns, and we get a critical mass of products, the more complex use cases will emerge, hopefully with simple and intuitive user interfaces.

14.9.1 Combining Bluetooth Classic Audio with Bluetooth LE Audio

Bluetooth Classic audio implementations will not disappear in the short term. The majority of audio products in the market today use pre-5.2 chipsets which aren't upgradeable, and many products, from TVs to cars, have a life of ten years or more. For at least the next five years,

phones and most earbuds will support both Bluetooth Classic Audio and Bluetooth LE Audio. Where both support the same use case, such as with HFP and A2DP, it will be up to the implementation to decide which to use. In phones, that will largely be down to the individual manufacturer and the silicon implementation. Products with a wider variety of stacks and open-source applications may be more diverse. Developers may need to put pressure on their silicon and stack providers to ensure they have access to the Bluetooth LE Audio features they need if they want to push the limits of their imaginations. It is important that the new opportunities are not constrained by a desire to keep things the way they have been for the last twenty years.

The important point for developers is to make sure it works for the user. Whilst CAP includes handover procedures for moving from unicast to broadcast, these only apply where both are Bluetooth LE Audio. In the real world, it's equally possible that the transition may be from A2DP to LE broadcast and then back to LE unicast or classic Bluetooth. For product developers, the rule must be to anticipate and allow any combination. The Bluetooth SIG runs regular unplugfests where developers can test their products with those from other manufacturers. As products start to incorporate more and more of the new features of Bluetooth LE Audio, these will be invaluable to ensure an interoperable ecosystem and a degree of uniformity in user experience.

14.9.2 Concentrate on understanding broadcast

One of the learnings from developing the specification is how difficult the concepts of broadcast are for anyone who is used to the peer-to-peer model of HFP and A2DP. Whilst unicast includes a lot of new concepts (see Chapter 3 to review them), the acknowledged packet model is relatively familiar. Broadcast, with a transmitter that is unaware of whether anyone is listening to it, and receivers which act totally independently and have no state machine, have been surprisingly challenging for many to take in. Adding BASS, alongside the Broadcast Assistant and the delegation of scanning, does seem to be a challenge to that orthodox thinking.

The examples in this chapter try to illustrate the flexibility which is allowed. Developers need to understand the limitations that are imposed by broadcast – that each BIG has a fixed structure for all BISes, which may mean there are times when multiple BIGs are more efficient. Scanning and filtering broadcasts is all important for a good user experience, so the relevant fields need to be supported, and where appropriate, configurable by users. They must understand the Basic Audio Announcement, the use of Targeted and General Announcements by Initiators and Acceptors and the meaning of the BASE structure. The interaction between Broadcast Assistants and Scan Delegates and the use of the characteristics in BASS are fundamental to the sharing and authentication processes described above.

The Bluetooth SIG has published a series of “How to” articles covering each of the main Auracast™ products – Transmitter, Receiver and Assistant, as well as an overview of how they fit together. These are useful starting points for anyone embarking on designing products.

Section 14.9 - Market development and notes for developers

Finally, anyone designing with Bluetooth LE Audio should take time to understand and implement the control features and appreciate the fact that they can be distributed between multiple different devices. These are all simple Client-Server relationships, but the contents of each of the services are very comprehensive and make the difference between a rich or a basic user experience. The fact that they are GATT based means that many of them can be implemented on existing phones.

Although it is a basic LE feature, developers should also make sure they understand the role of notifications. In Bluetooth LE Audio, notifications are the means whereby Servers ensure that everything within the topology is up to date. Knowing when to expect them, and what to do if they don't arrive when expected (which is normally to go and read the characteristic) is vital to provide a robust and intuitive user experience, where a user can pick and choose multiple products from different manufacturers and have confidence that they will all work with each other.

14.9.3 Balancing quality and application

Developers should not forget the difference between Optional and Mandatory in the Bluetooth LE Audio specifications. In many of the specifications, very little is mandated. Many features are mandated to be supported, such as the mandated codec settings in BAP, GMAP and TMAP, but that doesn't mean that an application needs to use them. BAP is designed to ensure that there is a lowest common level of interoperability at a good audio quality. The mandate is that if an application chooses to use them, they must be supported. If they're only optional and an application on an Initiator wants to use them, the Acceptor can reject that request. It means that there is a lot of flexibility in what a product can do.

Having said, that, going off-piste may impinge on interoperability. As an example, the QoS settings in BAP have been thoroughly tested and developers can be sure that every silicon supplier will have made sure their implementation is interoperable, so they should be used. However, there will be times when physics gets in the way, typically if you want to transmit multiple Bluetooth LE Audio Streams, where you will begin to run out of airtime. We know from a history of audio development that consumers don't necessarily want the highest quality – they want ease of use. CDs and streaming services became successful because they were easy to use, and the audio quality was adequate. The companies that invented them understood that by being brave and taking a step back from the audio quality treadmill, they could provide an experience that allowed them to win customers' hearts. Bluetooth LE Audio has potential for new, compelling services, especially with Auracast™, which should be designed with that in mind. The audio quality is there when it is needed, but so is a lot else.

14.9.4 Reinventing audio

In the last few decades, the way we consume audio has become concentrated into the hands of phone manufacturers and streaming services. What actually renders it, notwithstanding the inordinate success of Apple's AirPods and their competitors, is largely the minor partner.

Hearables have added neat features, but they haven't really played any major part in the development of the audio chain – they remain a passive peripheral to the phone. Bluetooth LE Audio's new topologies and distributed control features provide the potential for a new generation of innovation, where devices we have yet to imagine become an important part of our lives. At that point hearables may evolve to become the dominant partner in the audio experience. The aim of everyone involved in the Bluetooth LE Audio development has been to provide the tools for a new generation of innovation. I hope this book has inspired you to think outside the box and consider how that can be achieved.

Chapter 15. Glossary and concordances

In this final chapter, I've listed all of the specifications which comprise Bluetooth® LE Audio, the acronyms used in this book, along with a list of all of the Bluetooth LE Audio procedures and sub-procedures, along with where they are defined. I can only hope to provide an overview of the specifications within this book. These cross-references should help you navigate your way through the specifications.

15.1 Abbreviations and initialisms

The following abbreviations and initialisms are used in this book and throughout the Bluetooth LE Audio specifications.

Acronym / Initialism	Meaning
3GPP	Third Generation Partnership Project
A2DP	Advanced Audio Distribution Profile
ACAD	Additional Controller Advertising Data
ACL	Asynchronous Connection-oriented link
ACL	Asynchronous Connectionless
AD	Advertising Data
AdvA	Advertiser Address
AICS	Audio Input Control Service
ASCS	Audio Stream Control Service
ASE	Audio Stream Endpoint
ATT	Attribute Protocol
AVDTP	Audio Video Distribution Transport Protocol
AVRCP	Audio Video Remote Control Profile
BAP	Basic Audio Profile
BAPS	The set of BAP, ASCS, BASS and PACS
BASE	Broadcast Audio Source Endpoint
BASS	Broadcast Audio Scan Service
BAU	Broadcast Audio URI
BFI	Bad Frame Indication
BGR	Broadcast Gaming Receiver
BGS	Broadcast Gaming Source
BIG	Broadcast Isochronous Group
BIS	Broadcast Isochronous Stream
BMR	Broadcast Media Receiver
BMS	Broadcast Media Sender
BN	Burst Number
BR/EDR	Basic Rate/Enhanced Data Rate

Acronym / Initialism	Meaning
BW	Bandwidth
CAP	Common Audio Profile
CAS	Common Audio Service
CCID	Content Control Identifier
CCP	Call Control Profile
CG	Call Gateway
CIE	Close Isochronous Event
CIG	Connected Isochronous Group
CIS	Connected Isochronous Stream
CMAC	Cipher-based Message Authentication Code
CRC	Cyclic Redundancy Check
CSIP	Coordinated Set Identification Profile
CSIPS	The set of CSIP and CSIS
CSIS	Coordinated Set Identification Service
CSS	Core Specification Supplement
CT	Call Terminal
CTKD	Cross-Transport Key Derivation
CVSD	Continuous Variable Slope Decode
DCT	Discrete Cosine Transform
DECT	Digital Enhanced Cordless Telecommunications
DSP	Digital Signal Processor
DUN	Dial-up Networking
EA	Extended Advertisement
EATT	Enhanced ATT
EIR	Extended Inquiry Response
FB	Full Band (20 kHz audio bandwidth)
FT	Flush Timeout
GAF	Generic Audio Framework
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GC	Group Count
GMAP	Gaming Audio Profile
GMAS	Gaming Audio Service
GMCS	Generic Media Control Service
GPS	Global Positioning System
GSS	GATT Specification Supplement
GTBS	Generic Telephone Bearer Service
HA	Hearing Aid (as in the role described in the Hearing Access Profile)

Section 15.1 - Abbreviations and initialisms

Acronym / Initialism	Meaning
HAP	Hearing Access Profile
HARC	Hearing Aid Remote Controller
HAS	Hearing Access Service
HAUC	Hearing Aid Unicast Client
HCI	Host Controller Interface
HDMI	High-Definition Media Interface
HFP	Hands-Free Profile
HQA	High Quality Audio
IA	Identity Address
IAC	Immediate Alert Client
IAS	Immediate Alert Service
INAP	Immediate Need for Audio related Peripheral
IRC	Immediate Repeat Count
IRK	Identity Resolving Key
ksps	kilo samples per second
L2CAP	Logical Link Control and Adaption Protocol
LC3	Low Complexity Communication Codec
LD-MDCT	Low Delay Modified Discrete Cosine Transform
LE	Low Energy (as in Bluetooth Low Energy)
LL	Link Layer
LSB	Least Significant Bit
LSO	Least Significant Octet
LTK	Long Term Key
LTPF	Long Term Postfilter
LTV	Length Type Value
MAC	Message Authentication Code
MCP	Media Control Profile
MCS	Media Control Service
MDCT	Modified Discrete Cosine Transform
MEMS	Microelectromechanical System
MIC	Message Integrity Check
MICP	Microphone Control Profile
MICS	Microphone Control Service
MSB	Most Significant Bit
mSBC	Modified SBC (codec)
MSO	Most Significant Octet
MTU	Maximum Transmission Unit
NB	Narrow Band (4 kHz audio bandwidth)

Acronym / Initialism	Meaning
NESN	Next Expected Sequence Number
NPI	Null Payload Indicator
NSE	Number of Subevents
OOB	Out of Band
OTP	Object Transfer Profile
OTS	Object Transfer Service
PA	Periodic Advertisement
PAC	Published Audio Capabilities
PACS	Published Audio Capabilities Service
PAST	Periodic Advertising Synchronization Transfer
PBA	Public Broadcast Assistant
PBAS	Public Broadcast Audio Stream Announcement
PBK	Public Broadcast Sink
PBP	Public Broadcast Profile
PBS	Public Broadcast Source
PCM	Pulse Code Modulation
PDU	Protocol Data Unit
PHY	Physical Layer
PLC	Packet Loss Concealment
PSM	Protocol Service Multiplexer
PSM	Protocol/Service Multiplexer
PTO	Pre-Transmission Offset
QoS	Quality of Service
RAP	Ready for Audio related Peripheral
RFU	Reserved for Future Use
RSI	Resolvable Set Identifier
RTN	Retransmission Number
SBC	low-complexity Sub Band Codec
SDP	Service Discovery Protocol
SDU	Service Data Unit
SIRK	Set Identity Resolving Key
SM	Security Manager
SN	Sequence Number
SNS	Spectral Noise Shaping
SSWB	Semi Super Wide Band (12 kHz audio bandwidth)
SWB	Super Wide Band (16 kHz audio bandwidth)
TBS	Telephone Bearer Service
TMAP	Telephony and Media Audio Profile

Section 15.2 - Bluetooth LE Audio specifications

Acronym / Initialism	Meaning
TMAS	Telephony and Media Audio Service
TNS	Temporal Noise Shaping
UCI	Uniform Caller Identifier
UI	User Interface
uint48	unsigned 48-bit integer
UGG	Unicast Gaming Gateway
UGT	Unicast Gaming Terminal
UMR	Unicast Media Receiver
UMS	Unicast Media Sender
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
UUID	Universally Unique Identifier
UX	User Experience
VCP	Volume Control Profile
VCS	Volume Control Service
VOCS	Volume Offset Control Service
VoIP	Voice over IP
WB	Wide Band (8 kHz audio bandwidth)

Table 15.1 Acronyms and initialisms

15.2 Bluetooth LE Audio specifications

The following specifications comprise the Bluetooth LE Audio ecosystem. They build on new features which are present in the Core version 5.2 and above. The table lists the versions which were current when this book was published, and on which the text is based.

15.2.1 Adopted Specifications

These specifications have been adopted and implementations using them can be qualified.

Spec	Full Name	Version	Family
AICS	Audio Input Control Service	1.0	GAF
ASCS	Audio Stream Control Service	1.0.1	GAF
BAP	Basic Audio Profile	1.0.2	GAF
BASS	Broadcast Audio Scan Service	1.0	GAF
BAU	Broadcast Audio URI	1.0	Top level profile
CAP	Common Audio Profile	1.0	GAF
CAS	Common Audio Service	1.0	GAF
CCP	Call Control Profile	1.0	GAF

Spec	Full Name	Version	Family
CSIP	Coordinated Set Identification Profile	1.0.1	GAF
CSIS	Coordinated Set Identification Service	1.0.1	GAF
GMAP	Gaming Audio Profile	1.0	Top level Profile
GMAS	Gaming Audio Service (part of GMAP)	1.0	Top level Profile
GMCS	Generic Media Control Service (part of MCS)	1.0	GAF
GTBS	Generic Telephone Bearer Service (part of TBS)	1.0	GAF
HAP	Hearing Access Profile	1.0	Top level profile
HAS	Hearing Access Service	1.0.1	Top level profile
LC3	Low Complexity Communication Codec	1.0.1	Codec
MCP	Media Control Profile	1.0	GAF
MCS	Media Control Service	1.0.1	GAF
MICP	Microphone Control Profile	1.0	GAF
MICS	Microphone Control Service	1.0	GAF
PACS	Published Audio Capabilities Service	1.0.2	GAF
PBP	Public Broadcast Profile	1.0.1	Top level profile
TBS	Telephone Bearer Service	1.0	GAF
TMAS	Telephony and Media Audio Profile	1.0	Top level profile
TMAP	Telephony and Media Audio Service (part of TMAP)	1.0	GAF
VCS	Volume Control Service	1.0.1	GAF
VCP	Volume Control Profile	1.0	GAF
VOCS	Volume Offset Control Service	1.0.1	GAF

Table 15.2 *Adopted Bluetooth LE Audio Specifications*

15.3 Procedures in Bluetooth LE Audio

The following procedures are defined in the Bluetooth LE Audio specifications. Note that there are some cases where procedures have the same name. However, these are different procedures which are context sensitive.

Procedure or sub-procedure name	Specification	Section
Answer Incoming Call	CCP	4.4.13.1
ASE Control operations	BAP	5.6
ASE_ID discovery	BAP	5.3
Audio capability discovery	BAP	5.2
Audio data path removal	BAP	5.6.6.1
Audio data path setup	BAP	5.6.3.1
Audio role discovery	BAP	5.1
Available Bluetooth LE Audio Contexts discovery	BAP	5.4

Section 15.3 - Procedures in Bluetooth LE Audio

Procedure or sub-procedure name	Specification	Section
Broadcast Audio Stream configuration	BAP	6.3
Broadcast Audio Stream disable	BAP	6.3.3
Broadcast Audio Stream establishment	BAP	6.3.2
Broadcast Audio Stream Metadata update	BAP	6.3.3
Broadcast Audio Stream reconfiguration	BAP	6.3.1
Broadcast Audio Stream release	BAP	6.3.5
Broadcast Audio Stream state management	BAP	6.2
Call Control Point Procedures	CCP	4.4.13
CIS loss	BAP	5.6.8
Codec configuration	BAP	5.6.1
Configure Audio Input Description Notifications	VCP	4.4.3.8
Configure Audio Input State Notifications	VCP	4.4.3.1
Configure Audio Input Status Notifications	VCP	4.4.3.5
Configure Audio Location Notifications	VCP	4.4.2.3
Configure Audio Output Description Notifications	VCP	4.4.2.7
Configure Mute Notifications	MICP	4.4.1
Configure Volume Flags Notifications	VCP	4.4.1.3
Configure Volume Offset State Notifications	VCP	4.4.2.1
Configure Volume State Notifications	VCP	4.4.4.1
Coordinated Set Discovery procedure	CSIP	4.6.1
Disabling an ASE	BAP	5.6.5
Enabling an ASE	BAP	5.6.3
Fast Forward Fast Rewind	MCP	4.5.25
Join Calls	CCP	4.4.13.7
Lock Release procedure	CSIP	4.6.4
Lock Request procedure	CSIP	4.6.3
Move Call To Local Hold	CCP	4.4.13.3
Move Locally And Remotely Held Call To Remotely Held Call	CCP	4.4.13.5
Move Locally Held Call To Active Call	CCP	4.4.13.4
Move to First Group	MCP	4.5.39
Move to First Segment	MCP	4.5.29
Move to First Track	MCP	4.5.34
Move to Group Number	MCP	4.5.41
Move to Last Group	MCP	4.5.40
Move to Last Segment	MCP	4.5.30
Move to Last Track	MCP	4.5.35
Move to Next Group	MCP	4.5.38
Move to Next Segment	MCP	4.5.28

Procedure or sub-procedure name	Specification	Section
Move to Next Track	MCP	4.5.33
Move to Previous Group	MCP	4.5.37
Move to Previous Segment	MCP	4.5.27
Move to Previous Track	MCP	4.5.32
Move to Segment Number	MCP	4.5.31
Move to Track Number	MCP	4.5.36
Mute	VCP	4.4.1.6.7
Mute	VCP	4.4.3.7.3
Ordered Access procedure	CSIP	4.6.5
Originate Call	CCP	4.4.13.6
Pause Current Track	MCP	4.5.24
Play Current Track	MCP	4.5.23
QoS configuration	BAP	5.6.2
Read Audio Input Description	VCP	4.4.3.9
Read Audio Input State	VCP	4.4.3.2
Read Audio Input Status	VCP	4.4.3.6
Read Audio Input Type	VCP	4.4.3.4
Read Audio Location	VCP	4.4.2.4
Read Audio Output Description	VCP	4.4.2.8
Read Bearer List Current Calls	CCP	4.4.8
Read Bearer Provider Name	CCP	4.4.1
Read Bearer Signal Strength	CCP	4.4.5
Read Bearer Signal Strength Reporting Interval	CCP	4.4.6
Read Bearer Technology	CCP	4.4.3
Read Bearer UCI	CCP	4.4.2
Read Bearer URI Schemes Supported List	CCP	4.4.4
Read Call Control Point Optional Opcodes	CCP	4.4.14
Read Call Friendly Name	CCP	4.4.16
Read Call State	CCP	4.4.12
Read Content Control ID	MCP	4.5.44
Read Content Control ID	CCP	4.4.9
Read Current Track Object Information	MCP	4.5.12
Read Current Track Segments Object Information	MCP	4.5.11
Read Gain Setting Properties	VCP	4.4.3.3
Read Incoming Call	CCP	4.4.15
Read Incoming Call Target Bearer URI	CCP	4.4.10
Read Media Control Point Opcodes Supported	MCP	4.5.42
Read Media Information	MCP	4.5.1
Read Media Player Icon Object Information	MCP	4.5.2

Section 15.3 - Procedures in Bluetooth LE Audio

Procedure or sub-procedure name	Specification	Section
Read Media State	MCP	4.5.22
Read Mute	MICP	4.4.2
Read Next Track Object Information	MCP	4.5.14
Read Parent Group Object Information	MCP	4.5.18
Read Playback Speed	MCP	4.5.8
Read Playing Order	MCP	4.5.19
Read Playing Order Supported	MCP	4.5.21
Read Seeking Speed	MCP	4.5.10
Read Status Flags	CCP	4.4.11
Read Track Duration	MCP	4.5.4
Read Track Position	MCP	4.5.5
Read Track Title	MCP	4.5.3
Read Volume Flags	VCP	4.4.1.4
Read Volume Offset State	VCP	4.4.2.2
Read Volume State	VCP	4.4.1.1
Receiver Start Ready	BAP	5.6.3.2
Receiver Stop Ready	BAP	5.6.5.1
Relative Volume Down	VCP	4.4.1.6.1
Relative Volume Up	VCP	4.4.1.6.2
Released ASEs or LE ACL link loss	BAP	5.6.7
Releasing an ASE	BAP	5.6.6
Search	MCP	4.5.43
Set Absolute Track Position	MCP	4.5.6
Set Absolute Volume	VCP	4.4.1.6.5
Set Audio Input Description	VCP	4.4.3.10
Set Audio Location	VCP	4.4.2.5
Set Audio Output Description	VCP	4.4.2.9
Set Automatic Gain Mode	VCP	4.4.3.7.5
Set Bearer Signal Strength Reporting Interval	CCP	4.4.7
Set Current Group Object ID	MCP	4.5.17
Set Current Track Object ID	MCP	4.5.13
Set Gain Setting	VCP	4.4.3.7.1
Set Initial Volume	VCP	4.4.1.5
Set Manual Gain Mode	VCP	4.4.3.7.4
Set Members Discovery procedure	CSIP	4.6.2
Set Mute	MICP	4.4.3
Set Next Track Object ID	MCP	4.5.15
Set Playback Speed	MCP	4.5.9
Set Playing Order	MCP	4.5.20

Procedure or sub-procedure name	Specification	Section
Set Relative Track Position	MCP	4.5.7
Set Volume Offset	VCP	4.4.2.6.1
Stop Current Track	MCP	4.5.26
Supported Audio Contexts discovery	BAP	5.4
Terminate Call	CCP	4.4.13.2
Track Discovery - Discover by Current Group Object ID	MCP	4.5.16
Unmute	VCP	4.4.1.6.6
Unmute	VCP	4.4.3.7.2
Unmute/Relative Volume Down	VCP	4.4.1.6.3
Unmute/Relative Volume Up	VCP	4.4.1.6.4
Updating Metadata	BAP	5.6.4

Table 15.3 Procedures and sub-procedures defined in Bluetooth LE Audio specifications

15.4 Bluetooth LE Audio characteristics

The following characteristics are defined the Bluetooth LE Audio service specifications. The reference is to the table in the specification in which the characteristic is defined.

Characteristic	Specification	Table
ASE Control Point	ASCS	4.1
Audio Input Control Point	AICS	3.1
Audio Input Description	AICS	3.1
Audio Input State	AICS	3.1
Audio Input Status	AICS	3.1
Audio Input Type	AICS	3.1
Audio Location	VOCS	3.1
Audio Output Description	VOCS	3.1
Available Bluetooth LE Audio Contexts	PACS	3.5
Bearer List Current Calls	TBS	3.1
Bearer Provider Name	TBS	3.1
Bearer Signal Strength	TBS	3.1
Bearer Signal Strength Reporting Interval	TBS	3.1
Bearer Technology	TBS	3.1
Bearer Uniform Caller Identifier (UCI)	TBS	3.1
Bearer URI Schemes Supported List	TBS	3.1
BGR Features	GMAP	4.7
BGS Features	GMAP	4.7
Broadcast Audio Scan Control Point	BASS	3.1
Broadcast Receive State	BASS	3.1

Section 15.4 - Bluetooth LE Audio characteristics

Characteristic	Specification	Table
Call Control Point	TBS	3.1
Call Control Point Optional Opcodes	TBS	3.1
Call Friendly Name	TBS	3.1
Call State	TBS	3.1
Content Control ID	MCS	3.1
Content Control ID (CCID)	TBS	3.1
Coordinated Set Size	CSIS	5.1
Current Group Object ID	MCS	3.1
Current Track Object ID	MCS	3.1
Current Track Segments Object ID	MCS	3.1
Gain Setting Properties	AICS	3.1
GMAP Role	GMAP	4.7
Incoming Call	TBS	3.1
Incoming Call Target Bearer URI	TBS	3.1
Media Control Point	MCS	3.1
Media Control Point Opcodes Supported	MCS	3.1
Media Player Icon Object ID	MCS	3.1
Media Player Icon URL	MCS	3.1
Media Player Name	MCS	3.1
Media State	MCS	3.1
Mute	MICS	3.1
Next Track Object ID	MCS	3.1
Parent Group Object ID	MCS	3.1
Playback Speed	MCS	3.1
Playing Order	MCS	3.1
Playing Orders Supported	MCS	3.1
Search Control Point	MCS	3.1
Search Results Object ID	MCS	3.1
Seeking Speed	MCS	3.1
Set Identity Resolving Key	CSIS	5.1
Set Member Lock	CSIS	5.1
Set Member Rank	CSIS	5.1
Sink ASE	ASCS	4.1
Sink Audio Locations	PACS	3.2
Sink PAC	PACS	3.1
Source ASE	ASCS	4.1
Source Audio Locations	PACS	3.4
Source PAC	PACS	3.3
Status Flags	TBS	3.1

Characteristic	Specification	Table
Supported Audio Contexts	PACS	3.1
Termination Reason	TBS	3.1
TMAP Role	TMAP	
Track Changed	MCS	3.1
Track Duration	MCS	3.1
Track Position	MCS	3.1
Track Title	MCS	3.1
UGG Features	GMAP	4.7
UGT Features	GMAP	4.7
Volume Control Point	VCS	3.1
Volume Flags	VCS	3.1
Volume Offset Control Point	VOCS	3.1
Volume Offset State	VOCS	3.1
Volume State	VCS	3.1

Table 15.4 Characteristics defined in the Bluetooth LE Audio specifications

15.5 Bluetooth LE Audio terms

The following specific terms are defined and used throughout the Bluetooth LE Audio specifications. They are generally capitalised when used with their defined meanings. Where abbreviations, acronyms or initialism are used, these are indicated after the term.

Phrase	Specification	Section
Additional Controller Advertising Data (ACAD)	Core	Volume 6, Part B, Section 2.3.4.8
Application Profile	Core	Volume 1, Part A, Section 6.3
ASE identifier (ASE_ID)	ASCS	Section 4.1
ASE state machine	ASCS	Section 3
Audio Channel	BAP	Section 1.6
Audio Configuration	BAP	Section 4.4
Audio Location	BAP	Section 1.6 (and GA Assigned Numbers)
Audio Sink	BAP	Section 1.6 and 3.3
Audio Source	BAP	Section 1.6 and 3.3
Audio Stream Endpoint (ASE)	ASCS	Section 4.1
Broadcast Audio Source Endpoint (BASE)	BAP	Section 3.7.2.2
Broadcast Audio Stream	BAP	Section 1.6
Broadcast Isochronous Group (BIG)	Core	Volume 6, Part B, Section 4.4.6.2

Section 15.5 - Bluetooth LE Audio terms

Phrase	Specification	Section
Broadcast Isochronous Stream (BIS)	Core	Volume 6, Part B, Section 4.4.6.1
Broadcast Sink	BAP	Section 1.6
Broadcast Source	BAP	Section 1.6
Broadcast_ID	BAP	Section 3.7.2.1
BIG_Sync_Delay	Core	Volume 6, Part B, Section 4.4.6.4
Broadcast Game Receiver	BGR	Section 2
Broadcast Game Sender	BGS	Section 2
Call Control Client	CCP	Section 2
Call Control Server	CCP	Section 2
Call Gateway (CG)	TMAP	Section 2.2
Call Terminal (CT)	TMAP	Section 2.2
Caller ID	TBS	Section 1.9
CIG Identifier	Core	Volume 6, Part B, Section 4.5.14
CIG_Sync_Delay	Core	Volume 6, Part B, Section 4.5.14.1
CIS Identifier	Core	Volume 6, Part B, Section 4.5.13.1
Connected Isochronous Group (CIG)	Core	Volume 6, Part B, Section 4.5.14
Connected Isochronous Stream (CIS)	Core	Volume 6, Part B, Section 4.5.13
Context Type	PACS	Section 1.9 (and GA Assigned Numbers)
Coordinated Set	CSIP	Section 2.1
Enhanced ATT (EATT) bearer	Core	Volume 3, Part F, Section 3.2.1
Extended advertising (EA)	Core	Volume 6, Part B, Section 2.3.1
Generic Access Profile (GAP)	Core	Volume 3, Part C
Inband Ringtone	TBS	Section 1.9
Link Layer (LL)	Core	Volume 6, Part B
Local Retrieve	TBS	Section 1.9
Low Energy asynchronous connection (LE ACL)	Core	Volume 1, Part A, Section 3.5.4.6
Media Control Client	MCP	Section 2.1
Media Control Server	MCP	Section 2.1

Phrase	Specification	Section
PA_Interval	Core	Volume 6, Part B, Section 2.3.4.6
Packet Loss Concealment (PLC)	LC3	Appendix B
Periodic Advertising Synchronization Transfer (PAST)	Core	Volume 3, Part C, Section 9.5.4
Periodic Advertising Train (PA)	Core	Volume 6, Part B, Section 4.4.5.1
Presentation Delay	BAP	Section 7
Published Audio Capabilities (PAC) record	PACS	Section 2.2
Remote Broadcast Scanning	BAP	Section 6.5
Remote Hold	TBS	Section 1.9
Scan Offloading	BAP	Section 6.5
Service Data AD data type	CSS	Section 1.11
Service UUID	CSS	Section 1.1
Set Coordinator	CSIP	Section 2
Set Members	CSIP	Section 2
Silent Mode	TBS	Section 1.9
Sink ASE	ASCS	Section 4
Source ASE	ASCS	Section 4
SyncInfo	Core	Volume 6, Part B, Section 2.3.4.6
Unenhanced ATT bearer	Core	Volume 3, Part A, Section 10.2
Unicast Audio Stream	BAP	Section 1.6
Unicast Game Gateway	UGG	Section 2
Unicast Game Terminal	UGT	Section 2

Table 15.5 Defined terms in the Bluetooth LE Audio specifications

Index

- ACAD 124, 235
- Acceptor 57, 86, 170, 178
- ACL link loss 215
- Additional Controller Advertising Data
..... 124
- ADV_EXT_IND 123
- Advertising Set ID 128, 248
- AICS 48, 283
- AirPods 19
- Airtime 60, 100, 156
- Anchor Point 88, 99, 120
- Announcements 66
- ASCS 46, 177, 190
- ASE Control Point 195, 207
- ASE state machine 194
- ASE_ID 190, 207
- ASHA 22
- Assisted Listening Stream 319, 337, 348
- Audio Announcement 237
- Audio Channel 58
- Audio Configuration 159, 160, 310
- Audio Data Path 208
- Audio Input Control Service .. 48, 283, 289
- Audio Location 185, 246, 289
- Audio quality 92, 157
- Audio Sink 48, 181
- Audio Stream Configuration Service ... 177
- Audio Stream Control Service 46, 190
- Audio Stream Endpoint 190
- Audio_Active_State 319
- Audio_Channel_Allocation .. 127, 238, 337
- Audio_Channel_Counts 193
- Audio_Channel_Location 74
- Auracast™ 325
- Auracast™ Simple Transmitter Best
Practices Guide 330
- Automatic Gain Control 291
- Autonomous Operation 215
- AUX_ADV_IND 123, 233
- AUX_SYNC_IND 238
- Auxiliary Pointer 122
- Availability 65
- Available Audio Contexts 65, 187
- Available Audio Contexts characteristic 64
- Available_Source_Contexts 218
- BAIRF 234, 347
- BAP 46, 177, 194
- BAP Broadcast Audio Stream
Establishment 235
- BASE 126, 129, 226, 233
- Basic Audio Announcement 67
- Basic Audio Announcement Service ... 124
- Basic Audio Profile 46, 177, 194
- BASS 46, 131, 221, 241
- BAU 320
- Bidirectional CIS 101
- BIG 59, 109
- BIG Synchronization Point 119
- BIG_Offset 126
- BIG_Sync_Delay 135
- BIGInfo 111, 124, 233, 235, 238
- BIS 84
- BIS Spacing 125
- Bitrate 156
- BN 93
- Bragi 19, 30
- Broadcast Assistant 132, 223, 236, 241,
242, 243, 333, 366
- Broadcast Audio Announcement 67
- Broadcast Audio Announcement Service
..... 226, 237
- Broadcast Audio Immediate Rendering
Flag 234, 305, 347
- Broadcast Audio Reception Ending
procedure 172
- Broadcast Audio Reception Start
procedure 172, 225
- Broadcast Audio Reception Stop
procedure 225
- Broadcast Audio Scan Control Point .. 242
- Broadcast Audio Scan Control Point
characteristic 242, 322
- Broadcast Audio Scan Service 46, 221, 241
- Broadcast Audio Scan Service UUID .. 250
- Broadcast Audio Source Endpoint 126
- Broadcast Audio Start procedure 172, 224
- Broadcast Audio Stop procedure 172, 225
- Broadcast Audio Stream configuration
procedure 224
- Broadcast Audio Stream disable

procedure.....	224	CIG reference point	104
Broadcast Audio Stream establishment		CIG state machine	104, 206
procedure.....	224	CIG synchronization point	104
Broadcast Audio Stream Metadata update		CIG_Sync_Delay	135
procedure.....	224	CIS.....	59, 84, 88
Broadcast Audio Stream reconfiguration		Close Isochronous Event	92
procedure.....	224	Codec Configuration procedure.....	199
Broadcast Audio Stream release		Codec Configuration Settings	147
procedure.....	224	Codec Configured state.....	191, 193
Broadcast Audio Streams	221	Codec Specific Capabilities.....	74, 179, 200
Broadcast Audio Update procedure ...	172, 224	Codec Specific Configuration	200, 228
Broadcast Audio URI.....	320	Codec_Frame_Blocks_Per_SDU.....	74
Broadcast Game Receiver	313	Codec_ID.....	178
Broadcast Game Sender.....	313	Commander ...	80, 129, 170, 223, 240, 242, 298
Broadcast Isochronous Group	59, 109	Common Audio Profile .	51, 167, 170, 177
Broadcast Isochronous Stream.....	84, 107	Connected Isochronous Group.....	59
Broadcast Isochronous Terminate		Connected Isochronous Stream	59, 84, 88
procedure.....	236	Content Control ID.....	173, 265
Broadcast Receive State	242	Context Types	60
Broadcast Receive State characteristic	242, 248, 255, 336	Control Subevent.....	108, 118
Broadcast Receiver	222, 223	Coordinated Set.....	69, 168, 241, 258
Broadcast Source State Machine	224	Coordinated Set Identification Profile..	51, 69, 167
Broadcast to Unicast Audio Handover		Coordinated Set Identification Service..	51
procedure.....	172	Coordinated Set Member Discovery ...	173
Broadcast to Unicast Handover	353	Coordination Control.....	51
Broadcast Transmitter.....	223, 326, 367	CSIP	51, 69
Broadcast_Code...130, 231, 241, 250, 258,		CSIS.....	51, 69, 168
259, 367, 383		CSSN.....	108
Broadcast_ID	128, 248	CSTF	108
Broadcast_Name.....	319, 332	CVSD.....	140
Burst Number.....	74, 93, 95, 112	Disabling state	213
Call Control ID	68	EHIMA	11
Call Control Profile.....	50, 264	Enabling an ASE.....	206
Call Gateway	307	Encryption	231
Call State characteristic	268	Ending a unicast stream.....	212
Call Terminal	307	Enhanced ISOAL	306
CAP	51, 167, 170, 177	Extended Advertisements	225
CCID	68, 173	Extended Advertising.....	121, 233
CCP	50, 264	Extended Attribute Protocol	44
Change Microphone Gain Settings		failed calls	273
procedure.....	173	Flush Point.....	93
Change Volume Mute State procedure	173	Flush Timeout	93, 118
Change Volume Offset procedure.....	173	Frame Length	149
Change Volume procedure.....	173	Frame Size.....	146
Change_Counter	286	Framing.....	98, 153, 201
Channel Allocation	72	Frequency hopping.....	91
CIG	59	FT	93

Gain.....	291	52
Gaming Audio Profile.....	310	Low Latency.....	152
General Announcement.....	66	Made for iPhone	22
Generic Audio Framework.....	45	Max_Transport_Latency.....	153, 231
Generic Media Control Service.....	50, 265	Maximum Transmit Latency	201
Generic Telephone Bearer Service	50, 264	Maximum_SDU_Size.....	153
GMAP.....	310	MCP	50, 264, 275
GMCS	50, 265	MCS.....	50, 264, 275
Group Count	112	MCS state machine	275
Groups.....	276	Media Control Client.....	275
GTBS	50, 264	Media Control Profile.....	50, 264
HABS.....	303	Media Control Service.....	50, 264, 275
Handover.....	261	MEMS microphone.....	28
HARC	303	Metadata	131, 179, 207, 228, 246, 249
HAUC.....	303	MICP.....	49
Hearing Access Service	302	Microphone control.....	295
High Quality Public Broadcast Audio	331	Microphone Control Profile.....	49
High Reliability	152	Microphone Mute State procedure	173
Identifier Data pair	323	MICS.....	49
Immediate Need for Audio related		Missing Acceptors.....	216
Peripheral.....	174	Missing set members	175
Immediate Repetition Count.....	113	Modify Broadcast Source procedure	244
INAP.....	174	mono	163, 239, 246
Inband ringtone.....	271	MP3.....	138
Incoming call.....	268, 271	mSBC.....	140
Initiator.....	57, 86, 170	Multi-channel.....	159
IRC	113	multiple volume controls.....	284
ISO PDU.....	89	Multiplex Sink.....	315
ISO_Interval.....	125, 233	multiplexed audio streams	162
ISOAL	98, 134	Multi-profile	55
Isochronous Adaptation Layer	134	Multisink Sink.....	314
Isochronous Interval	88	Mute	287
Isochronous Payloads.....	89	Near Field Magnetic Induction.....	24, 69
Isochronous Stream.....	58, 83	NFMI.....	24
Language.....	319	notifications	259
Latency.....	138, 149, 343	NSE.....	93
LC3.....	52, 137	Number of Subevents	93, 112
LE Create BIG Parameters	232	Object Transfer Service	50, 277
LE Create CIS command.....	107	Out of band ringtones.....	271
LE Remove CIG command.....	107	PAC record	177
LE Set CIG Parameters	202	Packet Loss Concealment.....	76, 148
LE_Create_BIG.....	120	Packing.....	203, 231
LE_Set_Extended_Advertising_Data	233	PACS.....	46, 177
LE_Set_Periodic_Advertising_Data...	233	Parental_Rating.....	319
LE_Terminate_BIG	121	PAST.....	129, 130, 132, 247
Link Layer ID	98, 108	PBP.....	317, 318
Locally Held.....	268	Periodic Advertising	124, 225, 233
Lock.....	51	Periodic Advertising Sync Transfer.....	247
Low Complexity Communications Codec		Periodic Advertising Synchronization	

Transfer.....	129, 130	Set Coordinator.....	168, 241
Periodic Advertising train.....	123	Set Identity Resolving Key.....	169
Playback speed.....	280	Set Member.....	168
Playing order.....	280	Set Member Lock.....	169
Playing Tracks.....	277	Set Member Rank.....	169
PLC.....	76, 148	SID.....	128
Preferred Audio Contexts.....	64, 181	Silent Mode.....	272
Preferred Retransmission Number.....	201	Sink ASE.....	190, 210
Presentation Delay.....	74, 119, 149, 153, 201, 226, 233, 305, 311, 316, 343, 349	Sink ASE state machine.....	191
Presets.....	303	Sink Audio Location.....	186
Pre-Transmission Offset.....	74, 113, 118	Sink Audio Locations characteristic.....	338
Primary Advertising Channels.....	121	Sink led journey.....	55
Program_Info.....	319	Solicitation.....	250
Proprietary links.....	259	Solicitation Requests.....	250
PTO.....	113	Source ASE.....	190, 210
Public Broadcast Announcement.....	318	Source ASE state machine.....	213
Public Broadcast Assistant.....	318	Source Audio Locations.....	186
Public Broadcast Profile.....	124, 240, 317	Source_ID.....	248
Public Broadcast Sink.....	318	Standard Quality Public Broadcast Audio	330
Public Broadcast Source.....	318	stream handover.....	172
Published Audio Capabilities Service ...	46, 177	Streaming Audio Contexts.....	64
QoS.....	150	Sub_Interval.....	125
QoS configuration procedure.....	202	Sub_Interval spacing.....	91
QoS Configured state.....	212	Subevent.....	90
QR code.....	320, 329, 356, 367	Subgroup.....	127, 229
Quality of Service.....	150	Supported Audio Context Types.....	65
Rank.....	51	Supported Audio Contexts.....	64, 186
RAP.....	174	Supported Codec Specific Capabilities	179
Ready for Audio related Peripheral.....	174	Supported_Audio_Channel_Counts ...	180
Receiver Start Ready.....	191	Supported_Frame_Durations.....	180
Receiver Stop Ready.....	191, 213	Supported_Max_Codec_Frames_Per_SD U.....	181
Releasing state.....	213	Supported_Octets_Per_Codec_Frame	180
Remote Broadcast Scanning.....	253	Supported_Sampling_Frequencies.....	179
Remote Control.....	80	Synchronisation Reference.....	135, 149
Remotely Held.....	268	Synchronization Point.....	76
Resolvable Set Identity.....	168	SyncInfo.....	123
Retransmission.....	149	Target Latency.....	200
Retransmission Number.....	153, 154	Target PHY.....	200
Retransmissions.....	151	Targeted Announcement.....	66, 174
Ringtone.....	63	TBS.....	263
Robustness.....	92, 112	TBS Call Control Point characteristic..	271
RTN.....	153, 154, 231	TBS state machine.....	267
Sampling Frequency.....	152	Telephone Bearer Service.....	263
Sampling Rate.....	143	Telephony and Media Audio Profile....	306
SBC.....	141	Terminating calls.....	273
Scan Delegator.....	241, 243	TMAP.....	306
SDU Interval.....	153, 204	Top level profiles.....	52

Track Position	277	Unicast to Broadcast Handover	
Tracks.....	276	procedure.....	353
Transport Delay	135, 139	Updating unicast metadata	211
Transport Latency.....	314	URI.....	269
True Wireless	23	VCP.....	48, 283
Unicast Audio Start procedure	171	VCS	48, 283
Unicast Audio Stop procedure	171	Virtual Broadcast Assistants.....	340
Unicast Audio Update procedure.....	171	VOCS.....	48, 283
Unicast Game Gateway.....	310	Volume Control Profile	48, 283
Unicast Game Terminal.....	310	Volume Control Service.....	48, 283, 285
Unicast Media Receiver.....	308	Volume Controller.....	241
Unicast Media Sender.....	308	Volume Offset Control Service	48, 283, 288
Unicast to Broadcast Audio Handover		Volume State characteristic	285
procedure.....	172		

ABOUT THE AUTHOR

Nick Hunn is the author of the *Fundamentals of Short-Range Wireless* – the first book to cover Bluetooth Classic, Wi-Fi, Zigbee and Bluetooth LE. He developed some of the very first Bluetooth products to come to market and has started a number of successful technology companies. Nick has been involved in Bluetooth since its inception and has helped author most of the major Bluetooth requirements documents, including those for Bluetooth LE and Bluetooth LE Audio. From 2013 to 2024, he chaired the Bluetooth Hearing Aid working group, which developed the concept of Bluetooth LE Audio and has participated in writing all of the Bluetooth LE Audio specifications, including the LC3 codec.

Nick regularly talks and reports on the audio industry. In 2014, he coined the word “Hearables” to describe the new generation of personal audio products. He has produced two major reports on the market for wearable devices, which correctly predicted the growth and outstanding success of personal Bluetooth audio. These, and many other articles on the market and accompanying technology can be accessed on his blog at www.nickhunn.com. Nick’s aim with this book is to demystify the complexity of the specifications and help readers understand the potential they bring to the future market for wearables.