Introducing Bluetooth® LE Audio

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

by Nick Hunn

First published January 2022

Paperback ISBN: 979-8-72-723725-0
Hardback ISBN: 979-8-78-846477-0

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. They are too numerous to list here, but they are recorded in each of the Bluetooth® documents. 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.

For taking the time to read through the drafts and coming back with detailed comments, I would particularly 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, has helped make this into a much better book than it would otherwise have been. I must also thank the Bluetooth SIG for their help and support.

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, as well as the final book from cover to cover multiple times, questioning everything, as well as putting up with the many hours I spent writing it.

Contents

Introduction

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® Low Energy 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 Low Energy.

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 13 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.

A few of the Bluetooth LE Audio specifications are not yet adopted, so I've relied on pre-publication drafts which the Bluetooth SIG has made public. The versions used for this edition are listed in Chapter 13.

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 2020. 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 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 businesses.

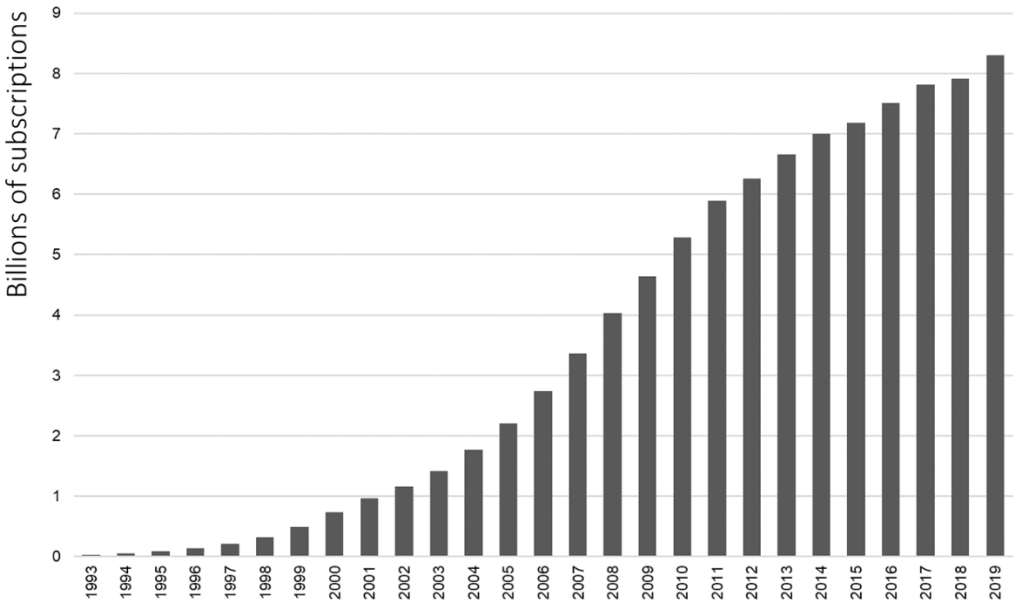


Figure 1.1 Growth of global mobile phone subscriptions

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 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 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.

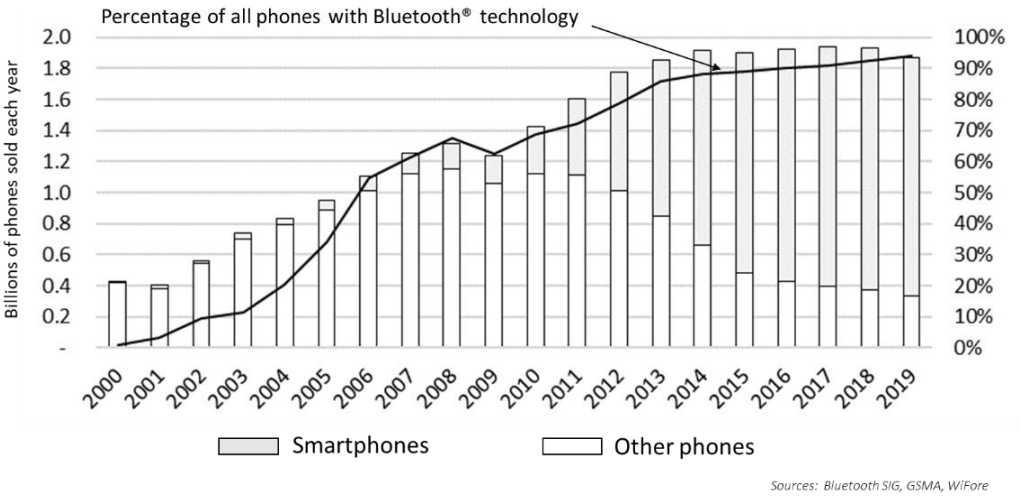


Figure 1.2 Percentage of mobile phones containing a Bluetooth® technology chip

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 to encourage more 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. 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 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 CDs, 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 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 coding 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.

Instead, 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 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. Subscribers flocked to the new offering. Six months later, it became even easier, with the launch of the first iPod.

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 initial mobile music market was one where you took your music with you, but didn't necessarily play it while you were 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, or 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.

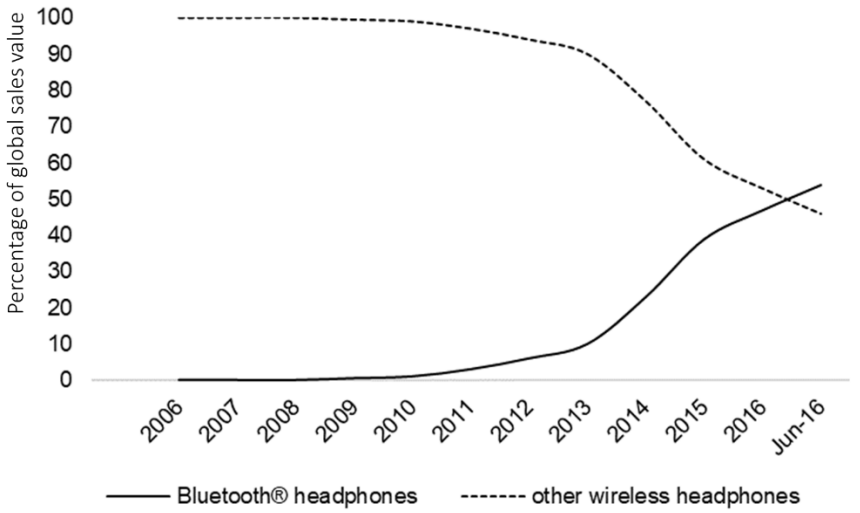


Figure 1.3 The transition from wired to Bluetooth® headphones

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.

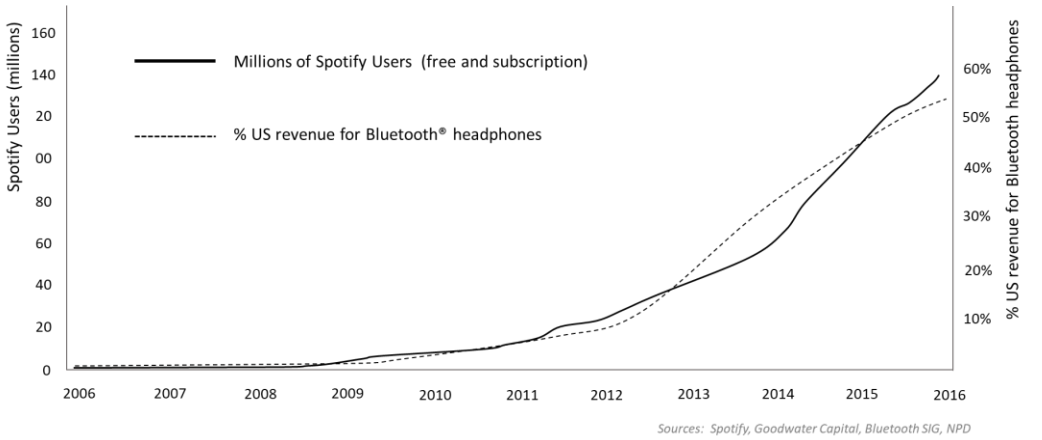


Figure 1.4 The growth of Spotify subscriptions and Bluetooth® headphones

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 Low Energy 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.

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 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. They're still making iconic designs, launching their third generation at the start of 2021. However, it was Bragi's Dash that really caught the imagination. In the spring of 2014, they broke the Kickstarter record for the highest funded campaign to 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.

Gradually, bigger names started to dip their toes into the hearables pond, utilizing their deeper resources to make these difficult products. 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 over 250 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. Despite supply chain problems, it is estimated that around 500 million earbuds were shipped in 2020, with a prediction that the number will double by the end of 2022. 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 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 reservations, 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 MEMS[1] 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 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 during the course of a day, constantly listening to and changing what they do with other

[1] 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.

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 legacy

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 hear audio and 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 Low Energy specification, 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 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, most of which have now been adopted, or are in the process of being adopted.

1.2 Limitations and proprietary extensions

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

In 2014, Apple launched its own proprietary Bluetooth Low Energy 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 aids 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.

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. 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. However, it is likely that most new development will move towards the globally interoperable Bluetooth LE Audio standard.

1.2.2 True Wireless

A practical solution to send 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 the success of Apple's Airpods, which use Apple's own, proprietary chipset, Qualcomm provided 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 synchronisation, assuming that each audio sink could make up its own mind about when to render the stream. 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 came up with is known as the replay or forwarding approach, as shown in Figure 1.5.

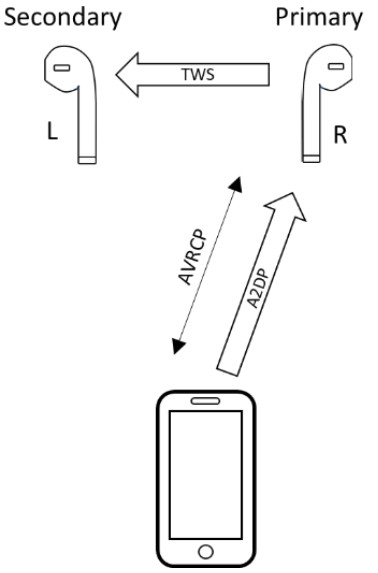


Figure 1.5 Replay scheme for TWS

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 which gets around the absorption problem. 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.

As the primary device 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 most 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

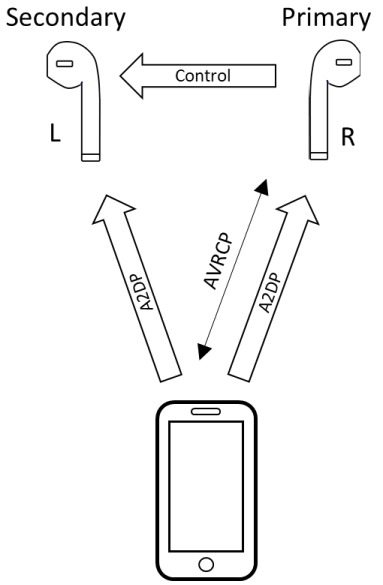


Figure 1.6 The sniffing approach for TWS

Here, the added latency of the relay approach is removed by both earbuds listening 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 it with that information, 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

Sharing audio isn't just about sending signals to a pair of earbuds and hearing aids – it's also about extending the number of people who can listen to the signal. Whilst public installations can cater for large numbers of people listening simultaneously, there's a more personal application where you want to share your music with a friend. The White Paper on multiple headphones (referred to above) sets out how shared listening with one other person is possible with A2DP, but the solution it suggests never seems to have made it into products. Instead, this segment of the market has largely been owned by Tempow – a Paris based Bluetooth software company. They have developed a phone based audio operating system which they call Dual-A2DP, which generates a separate left and right synchronised stream, allowing two separate pairs of earbuds to render the streams at the same time. Although this is useful, it is limited and has only been taken up by a few manufacturers. Bluetooth LE Audio goes beyond this by providing a scalable solution to transition from one to many listeners.

1.3 What's in a hearable?

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 also 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.

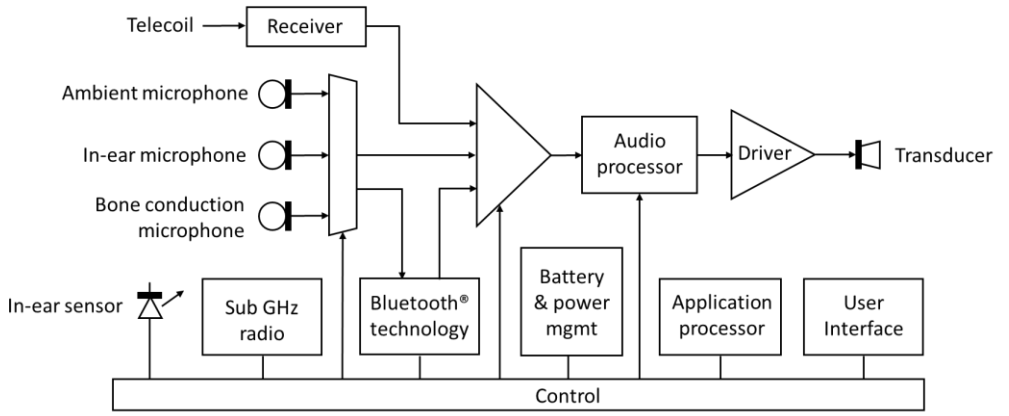


Figure 1.7 Architecture of a simple Bluetooth® hearing aid

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 one, these all need to be routed via a sub-GHz radio (normally NFMI), whilst the primary signal is buffered to synchronise 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 high 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 AirPod.

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 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.

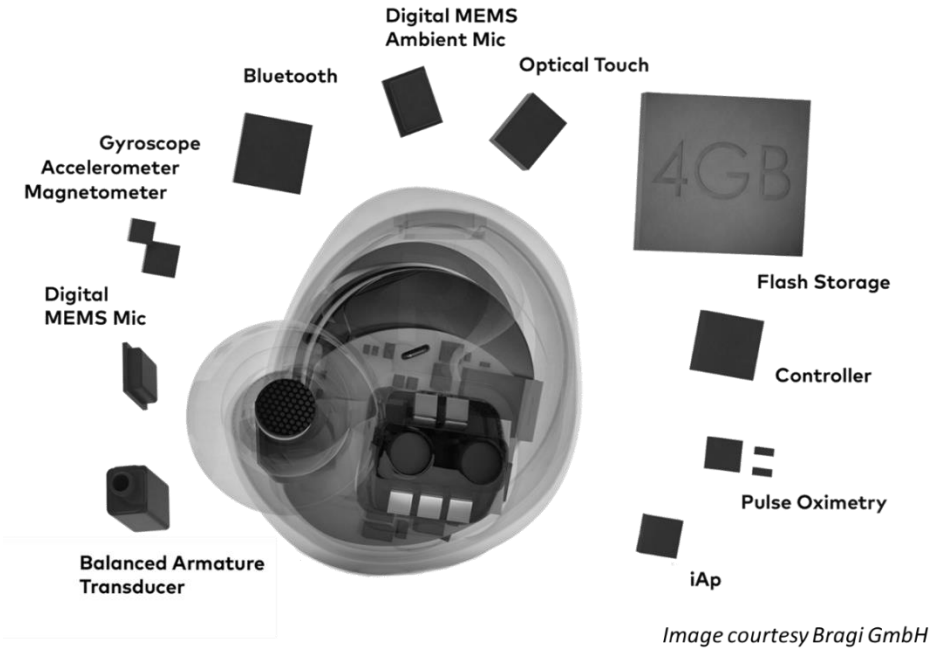


Figure 1.8 The Dash earbud from Bragi - the first real hearable

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 that 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 phone operating system. There is no guarantee that it will ever provide feedback which is compelling enough for the user to pay a monthly subscription to support the ongoing development costs. Without that insight, users give up, which is why so many fitness bands are now sitting at the back of bedroom drawers. It doesn't help that few companies making these devices have a data analytics business background, rather than a pure hardware business model.

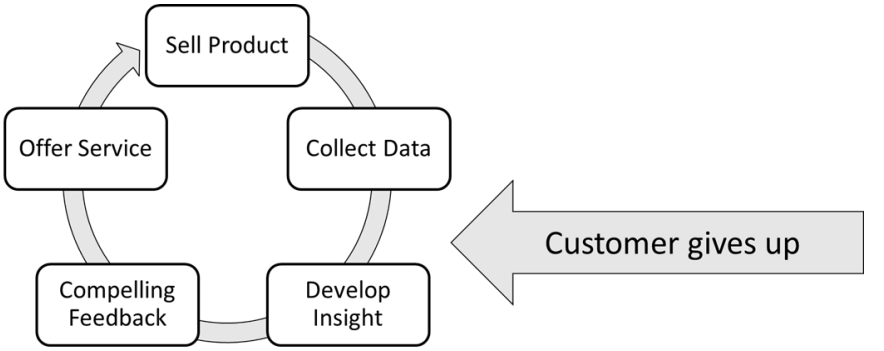


Figure 1.9 The Catch 22 of health and fitness data

The difficulty of developing an insight business model is why almost all of the 500 million earbuds which shipped in 2020 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 the wrist isn't. But the industry has learnt the lesson that data is also hard, which means that these sensors are likely to appear 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. 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.

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 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.

2.1.1 The hearing aid 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 .

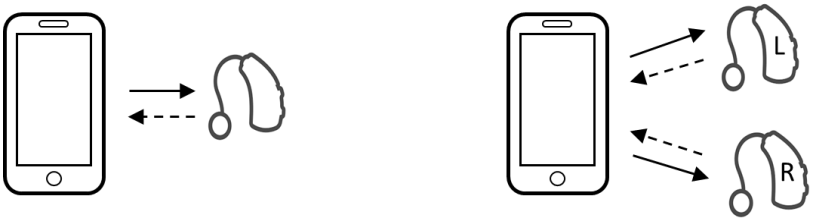


Figure 2.1 Basic Hearing Aid topologies

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 synchronised 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 within a hearing aid is minimal – less than a few milliseconds, this doesn't present a problem. However, it become a problem in a situation like that of Figure 2.2, where some of the wearer's family is 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.

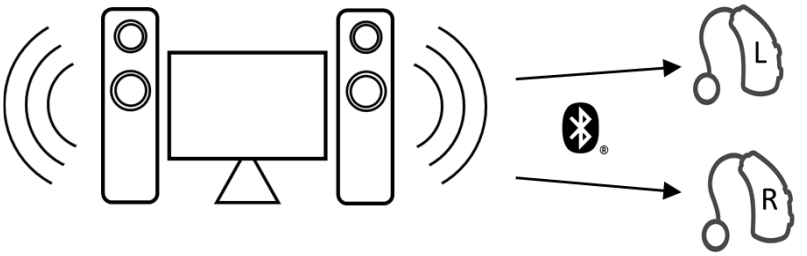


Figure 2.2 Bluetooth® LE Audio streaming with ambient sound

If the delay between the two audio signals is much more than 30 – 40 milliseconds, it begins to add 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.

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

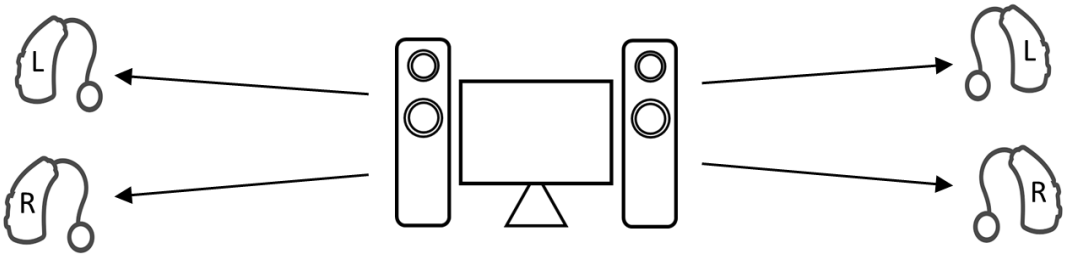


Figure 2.3 Adding in multiple listeners

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 or care home. That requirement lies on a spectrum which extends to the provision of a broadcast replacement for the current telecoil induction loops. 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.

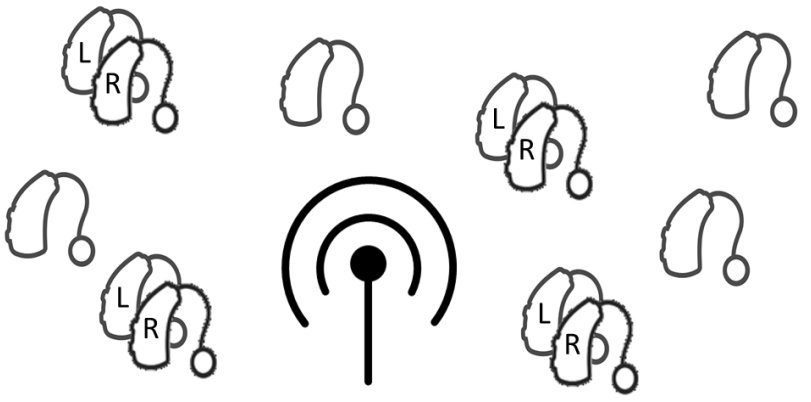


Figure 2.4 A broadcast topology to replace telecoil infrastructure

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, 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 what they are, 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, but others would. In environments like the home, you wouldn't want to pick up your neighbour's TV. 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.

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 level[2] 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 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 lower 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

[2] 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.

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. 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.

2.1.2 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 a second round of work to determine how best to meet the hearing aid 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 Low Energy. 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 the Core 4.1 Low Energy specification.

The proposal for the Core was to implement a new feature called Isochronous[3] Channels which could carry audio streams in Bluetooth LE, alongside an existing ACL[4] channel. The ACL channel would be used to configure, set up and control the streams, as well as carrying

[3] Isochronous means a sequence of events which are repeated at equal time intervals.
[4] ACL channels are Asynchronous Connection-oriented Logical transports which are the 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. An ACL link must always be present during the life of a CIS.

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 scheme[5] 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. With the requirements of Bluetooth LE Audio, 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 synchronise 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 Synchronisation Transfer, which is key to making broadcast acquisition simple. PAST is a really useful feature for hearing aids, as scanning takes a lot of power. Minimizing scanning in is a useful feature 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

[5] 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.

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.

The Extended Advertising features were adopted in the Core 5.1 release, with Isochronous Channels, EATT and ISOAL in the more recent Core 5.2 release, paving the way for all of the other Bluetooth LE Audio specifications to be built on top of them.

2.1.3 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 third 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.

2.1.4 Evolving beyond HFP and A2DP

The fourth round of requirements was a reflection that audio and telephony applications have outpaced HFP and A2DP. Many calls today are VoIP[6] 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 for HFP and A2DP. The new Bluetooth LE Audio architecture was going to have to go beyond that and incorporate multi-profile support by design, with robust and interoperable transitions between devices and applications, as well as between unicast and broadcast.

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 Chapter 12, was born.

Potential new use cases started to proliferate. If we could synchronise stereo channels for two earbuds, why not for surround sound? Companies were keen to make sure that it 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 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.

---

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

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 features 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.

2.2 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 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.

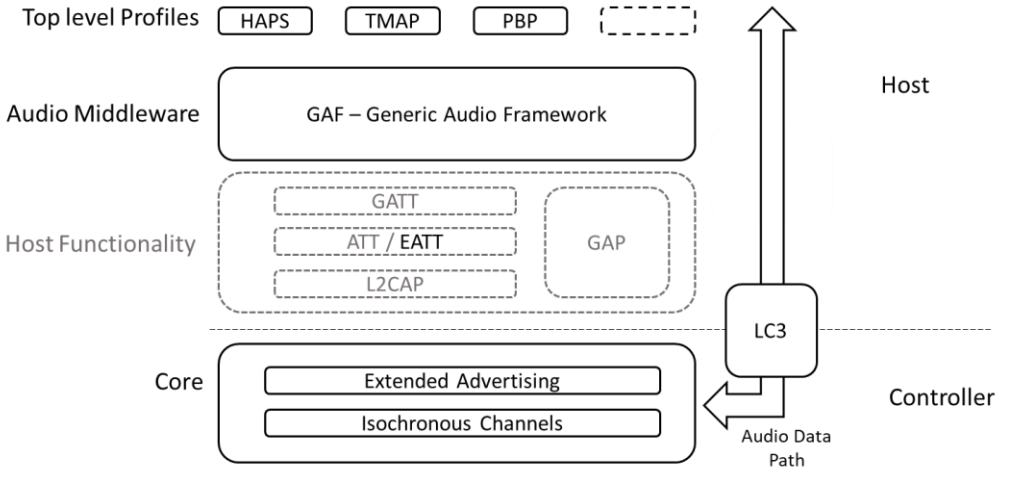


Figure 2.5 The Bluetooth® LE Audio architecture

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.

At first glance, the Bluetooth LE Audio architecture looks complex, as we've ended up with 23 different specifications inside the Generic Audio Framework, as well as an expanded Core and the new LC3 codec. But there's a logic to this. Each specification is trying to encapsulate the specific elements of the way you set up and control different aspects of an audio stream. In the rest 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.

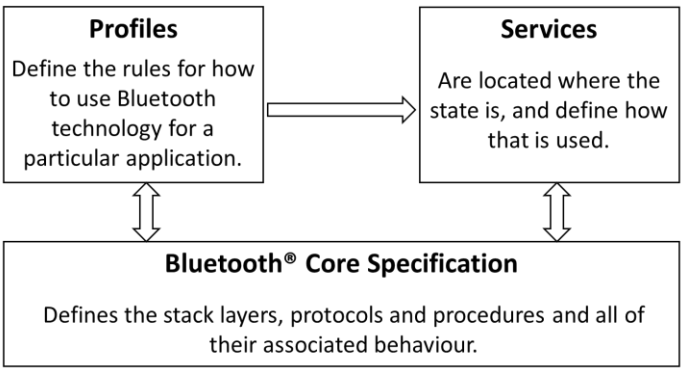


Figure 2.6 The Bluetooth® LE Profile and Service model

In Bluetooth LE, Profiles and Services can be considered as Clients and Servers. The Service is implemented where the state[7] 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.

[7] State refers to the value of a parameter or the current position within a state machine.

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 many-to-one 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 and acting as a Client. 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 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 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 four 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)[8] and the Voice Control Profile (VCP), one profile can operate on three different services. The Public Broadcast Profile (PBP) is an anomaly, as it's a profile without 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.

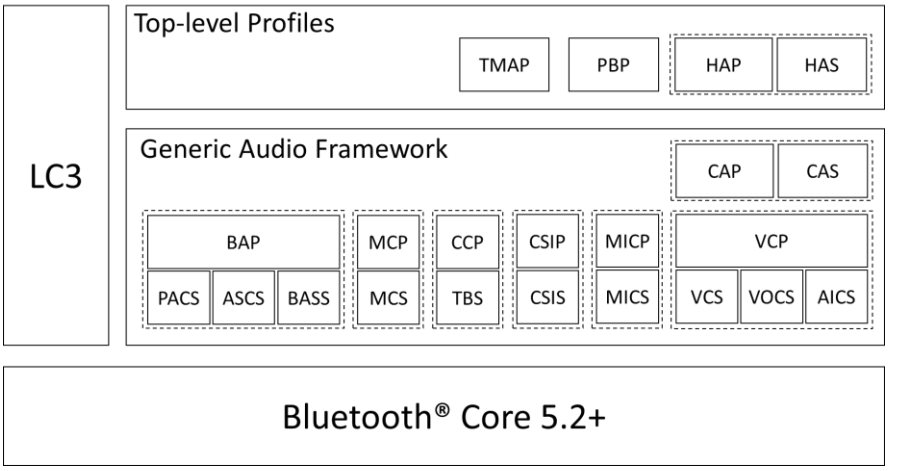


Figure 2.7 Overview of the Bluetooth LE Audio Specifications

[8] 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.

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 functional groups, arranged as in Figure 2.8.

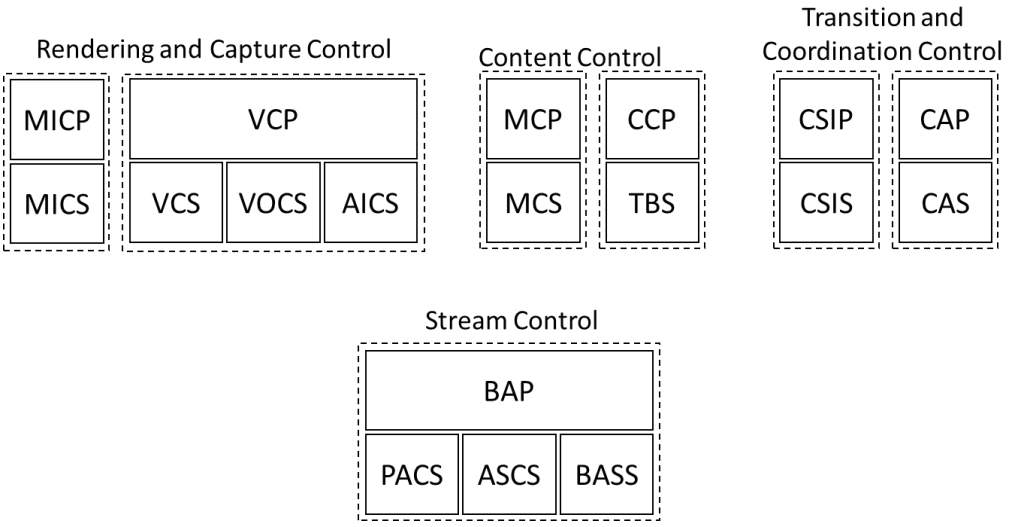


Figure 2.8 Functional grouping of specifications within the Generic Audio Framework

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 we set up the underlying Isochronous Channels which carry the audio data. 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.

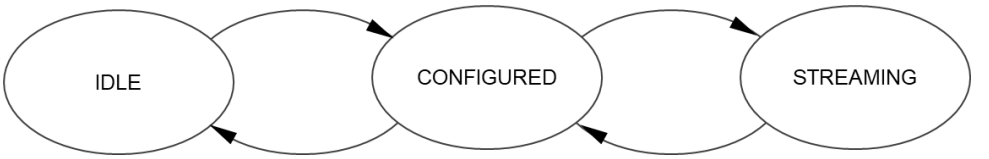


Figure.2.9. The simplified Isochronous Channel state machine

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, a state machine is only defined for the transmitter and 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 Audio, the final gain of the volume is performed in the hearing aid, earbud or speaker, and not on the incoming audio stream (although top level profiles may require that as well). 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 there are multiple Audio Streams, as with earbuds and hearing aids, a second service is required. VOCS - the Volume Offset Control Service, effectively acts as a balance control[9], 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. The Audio Input Control Service (AICS) acknowledges the fact that most devices have the ability to support a number of different audio streams, as illustrated in Figure 2.10. AICS provides the ability to control multiple different inputs that can be mixed together and rendered within your earbud or speaker. illustrates how these three services could be used in a soundbar which has a Bluetooth, HDMI and microphone input.

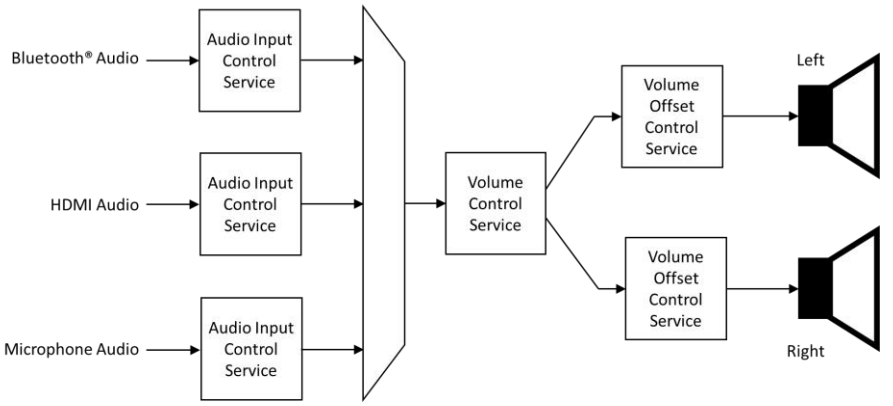


Figure 2.10 Audio Input Control Service (AICS), Volume Control Service (VCS) and Volume Offset Control Service (VOCS)

[9] "Balance" is the usual consumer nomenclature. Audio engineers would probably consider it to be a mixing control.

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 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 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. This ensures that they all have a synchronised 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.

MICP works in conjunction with AICS and MICS to control the overall gain and muting of multiple microphones. They 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 their use.

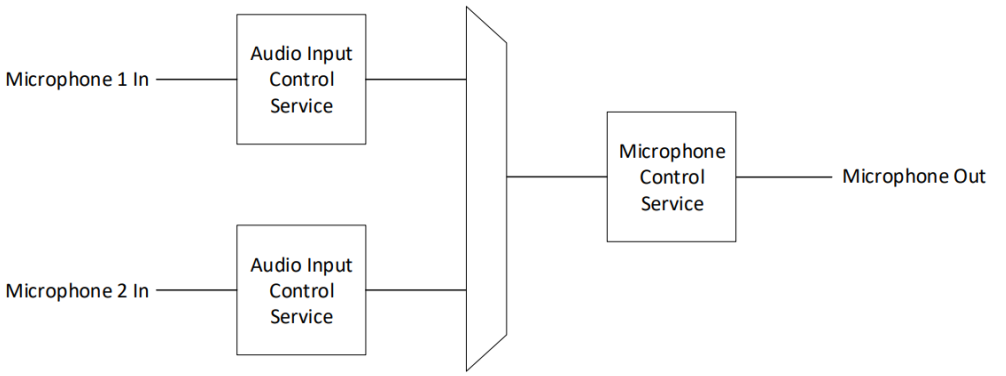


Figure 2.11 The Audio Input Control Service (AICS) used with the Microphone Control Service (MICS)

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 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 receiving device, 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 can be controlled from a smartwatch 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 accommodate the fact that we now use telephony in many different forms. It's no longer just traditional circuit switched[10] and cellular bearers, but PC and web-based communication

[10] Circuit switched is a definition derived from original telephone topology, where end-to-end connections were made by physically switching, using banks of relays. The term endured until IP

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 syncnronisation 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, 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.

---

routing technologies appeared.

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, allowing the design of ubiquitous, distributed remote control for audio. It is particularly useful for encrypted broadcasts, where it provides a way to convert broadcast transmissions into a private listening experience. We'll explore that in more detail in Chapter 8.

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 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. The first of 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)[11], which specifies the use of higher quality codec settings and more complex media and telephony control, and the Public Broadcast Profile (PBP), which helps users select globally interoperable broadcast streams. Public Broadcast Profile is 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 releases 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

[11] There is a corresponding, minimalist TMAS specification, which is included within the TMAP specification.

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 both 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 used ones, as a help to anyone 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	*aches (as in aches and pains)*
ASE	Audio Stream Endpoint	*ase (rhymes with case)*
ATT	Attribute Protocol	*at*
BAP	Basic Audio Profile	*bap (rhymes with tap)*
BAPS	The set of BAP, ASCS, BASS and PACS	*baps*
BASE	Broadcast Audio Source Endpoint	*base (rhymes with case)*
BASS	Broadcast Audio Scan Service	*rhymes with mass, not mace*
BIG	Broadcast Isochronous Group	*big*
BIS	Broadcast Isochronous Stream	*biss*
CAP	Common Audio Profile	*cap (rhymes with tap)*
CAS	Common Audio Service	*cas (rhymes with mass)*
CIG	Connected Isochronous Group	*sig*
CIS	Connected Isochronous Stream	*sis*
CSIP	Coordinated Set Identification Profile	*see - sip*
CSIS	Coordinated Set Identification Service	*see - sis*
CSIPS	The set of CSIP and CSIS	*see - sips*
EATT	Enhanced ATT	*ee - at*
GAF	Generic Audio Framework	*gaffe*
GAP	Generic Access Profile	*gap (rhymes with tap)*
GATT	Generic Attribute Profile	*gat (rhymes with cat)*
HAP	Hearing Access Profile	*hap (rhymes with tap)*
HARC	Hearing Aid Remote Controller	*hark*

Abbreviation	Meaning	Pronunciation
HAS	Hearing Access Service	*hass (rhymes with mass)*
HAUC	Hearing Aid Unicast Client	*hawk*
INAP	Immediate Need for Audio related Peripheral	*eye - nap*
L2CAP	Logical Link Control and Adaptation protocol	*el - two - cap*
MICP	Microphone Control Profile	*mick - pee*
MICS	Microphone Control Service	*mick - ess*
PAC	Published Audio Capabilities	*pack*
PACS	Published Audio Capabilities Service	*packs*
PAST	Periodic Advertising Sync Transfer	*past (rhymes with mast)*
PBAS	Public Broadcast Audio Stream Announcement	*pee - bass (bass rhymes with mass)*
PHY	physical layer	*fy (rhymes with fly)*
QoS	Quality of Service	*kwos*
RAP	Ready for Audio related Peripheral	*rap*
SIRK	Set Identity Resolving Key	*sirk (like the first syllable of circus)*
TMAP	Telephony and Media Audio Profile	*tee - map*
TMAS	Telephony and Media Audio Service	*tee - mas*
VOCS	Volume Offset Control Service	*vocks*

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, INAP, 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 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 core 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 would 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 Sink[12] 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, information from a 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.

[12] 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 be more involved in decisions.

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 is Figure 3.1.

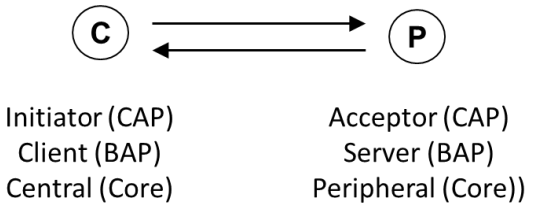


Figure 3.1 Bluetooth® LE Audio roles

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. There's one other abbreviation I'll make. As only Initiators can broadcast Audio Streams, I'll save a few words and refer to Initiators acting as Broadcast Sources as Broadcasters, unless there's a need to be explicit.

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.

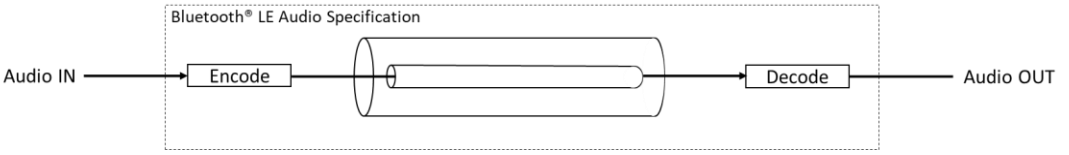


Figure 3.2 Bluetooth® LE Audio terminology

The diagram shows the conceptual transmission of audio data from left to right. At the very left, we have audio coming in, which is typically an analogue 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.) 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 specification[13].

How that audio input is carried to the audio output is what is defined inside the Bluetooth specifications. They are 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 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 synchronise 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 are illustrated in Figure 3.3.

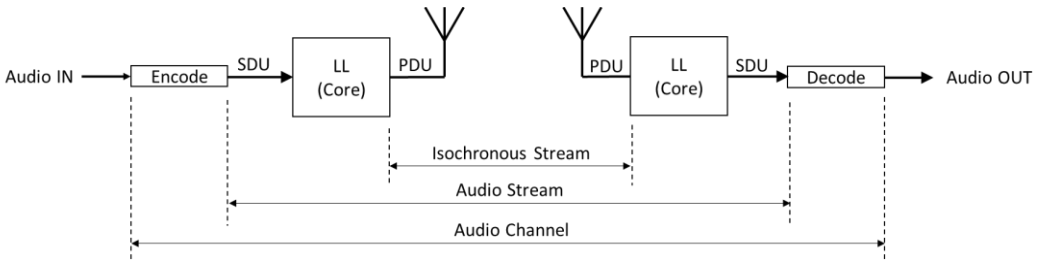


Figure 3.3 Representation of streaming terms

[13] With the exception of the Presentation Delay, which states when it is rendered. This is explained in Section 3.10.

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 rendering a left and a right stereo stream into two separate earbuds, with one or two return streams from their microphones, which 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.

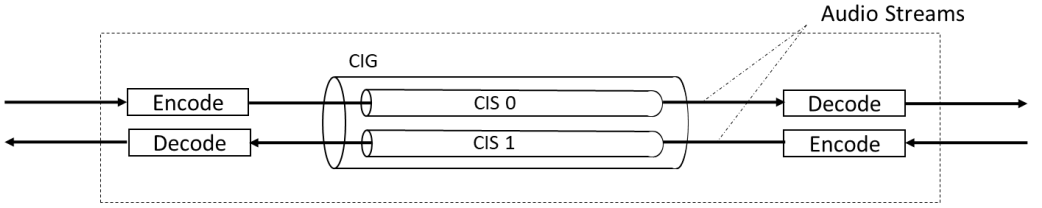


Figure 3.4 Two separate CISes in a CIG

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, carrying audio data in both directions in the same CIG. We'll see exactly how that works in the next chapter.

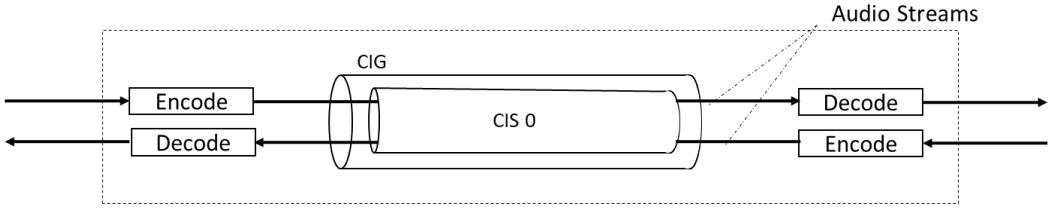


Figure 3.5 A bidirectional CIS in a CIG

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.

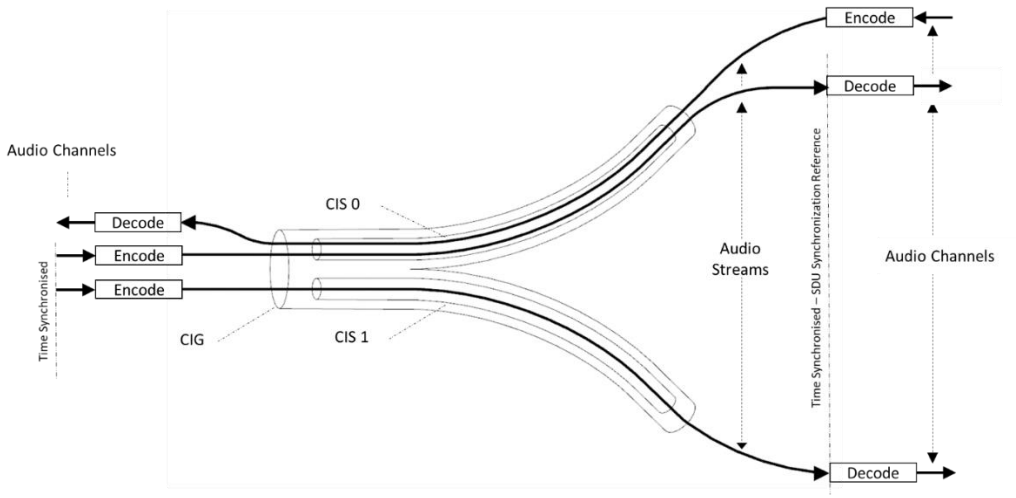


Figure 3.6 A unidirectional and bidirectional CIS for two Acceptors, e.g., a phone supporting a telephone call with two hearing aids.

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.

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 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 private (non-Bluetooth) conversation which they don't want interrupted, 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 restrict incoming calls to one specific phone, 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

Bit	Context Type	Description
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

Most of these are obvious, but two of the Context Types are worth special attention: «Ringtone» and «Unspecified»[14].

The «Ringtone» Context Type is used to announce an incoming phone call, but only where there is an inband[15] 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

[14] When Context Types are used in text, they are enclosed in guillemets, i.e. « and ».
[15] 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.

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 stream 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 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 is 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 land-line phone.

Context Types are used by both Initiators and Acceptors to provide information about the 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 LTV[16] 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.

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

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.

3.5 Availability

Bluetooth LE Audio supports far more possibilities and combinations than Bluetooth Classic Audio. 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 ones they're interested in participating in in the future. 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 Broadcasters, 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 Broadcasters, 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 synchronise to it, unless they have specifically set «Unspecified» as non-available.

3.6 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 left to the audio sink to interpret how it was rendered. Bluetooth LE Audio is intended to address applications with multiple speakers or earbuds. It has the ability to optimise airtime for each Audio Sink, by only sending it 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.

To accomplish this, devices need 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. These are defined in the Bluetooth Generic Audio Assigned Numbers and follow the categorization of CTA-861-G's Table 34 codes[17] for speaker placement. The values are expressed as bits in a four-octet wide bitfield. The most common Audio Locations are shown in Table 3.1.

Audio Location	**Value (bitmap)**
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)
Prohibited	0x00000000

Table 3.2 Common Audio Location values

Every Acceptor which can receive an audio stream must set at least one Audio Location – leaving the entire bitfield blank is not allowed. 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 or Front Right.

Note that mono is not a location, as mono is a property of the stream, not the physical rendering device. A single channel speaker would normally set both the Front Left and Front Right audio locations. An Initiator would determine the number of streams an Acceptor supports, along with the number of speakers available and make a decision of whether to send a downmixed stereo stream (i.e., mono), or a stereo stream that it assumes the speaker could downmix to mono. Broadcasters would denote a mono stream by labelling it as Front Left and Front Right. We'll look further at exactly how Audio Locations are used in Chapter 5, but it brings us on to Channel Allocation.

3.7 Channel Allocation (multiplexing)

You always knew what was being transported in the Bluetooth Classic Audio profiles. HFP carries a mono audio stream, and A2DP uses the four channel modes of SBC codec to transmit mono, dual channel, stereo or joint stereo encoded streams. Bluetooth LE Audio is a lot more flexible, allowing a CIS or BIS to contain one or more channels multiplexed into a single packet, limited only by the available bandwidth.

[17] https://archive.org/stream/CTA-861-G/CTA-861-G_djvu.txt

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 two separate left and right channels. That was due to its ability to encode differences between the two input audio channels. The more efficient design of LC3 means that there is very little advantage in joint stereo coding compared to encoding each channel separately and then concatenating the individual encoded frames. This approach allows more than two channels to be encoded and grouped together.

However, this meant that a mechanism needed to be introduced to package together multiple encoded frames for multiple channels, which is performed using Channel Allocation [BAP Section 4.2].

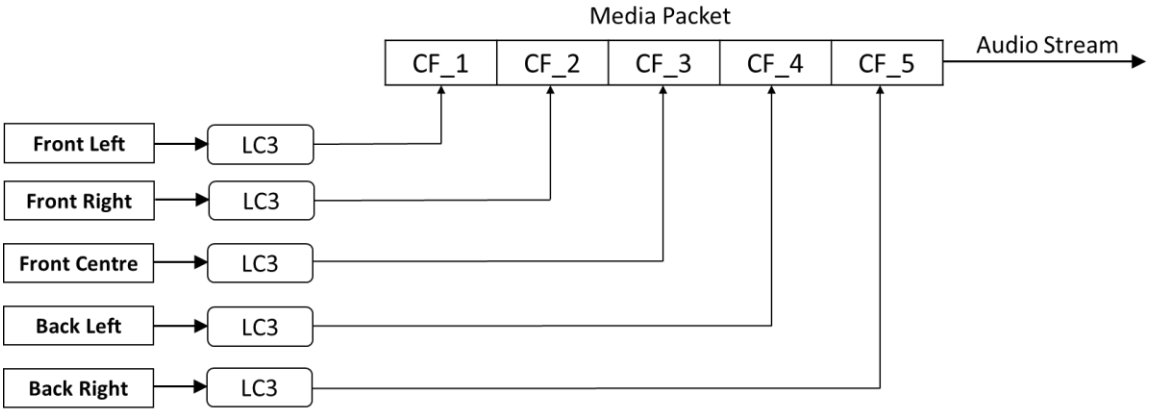


Figure 3.7 Example of multiplexing multiple Bluetooth® LE Audio Channels

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 3.2. 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).

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_N_1 refers to frames from the first sample of each of the incoming audio channels; CF_N_2 to those from the next samplings of those audio channels.

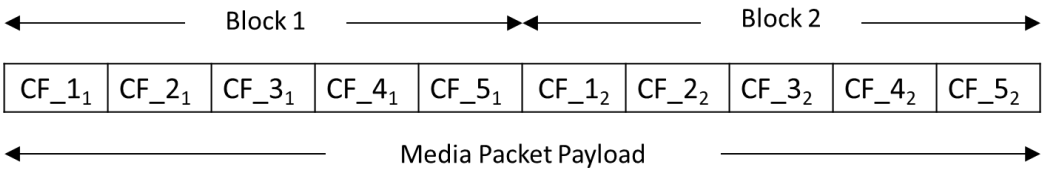


Figure 3.8 Media Packet containing two blocks of five Audio Channels

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 Audio_Channel_Location 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.

3.8 Call 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 content control signal, 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 Supplement[18] 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 3.12.

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

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. However, the flexibility of Bluetooth LE Audio, including the new use cases coming from broadcast, raised the need for a way to link Acceptors together as sets of devices, which 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 is down to the implementation. When you adjust the volume for a pair of earbuds, the volume of both should change at the same time[19], 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 CSIS), which are referenced from within CAP. The main feature of CSIS and CSIP, aside

[18] This is a document that describes generic features used within Bluetooth Low Energy.
[19] The Volume Control profile has the flexibility to allow these to be changed independently if the user prefers.

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, which is not significantly attenuated by the head, typically using 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 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 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 send 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

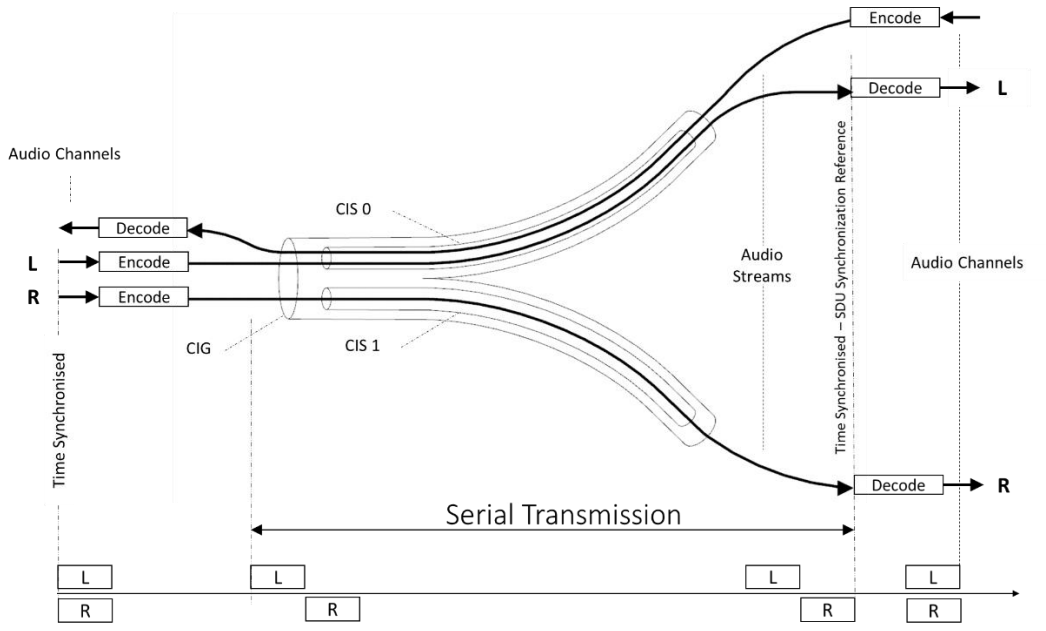


Figure 3.9 The serial transmission of audio data

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.

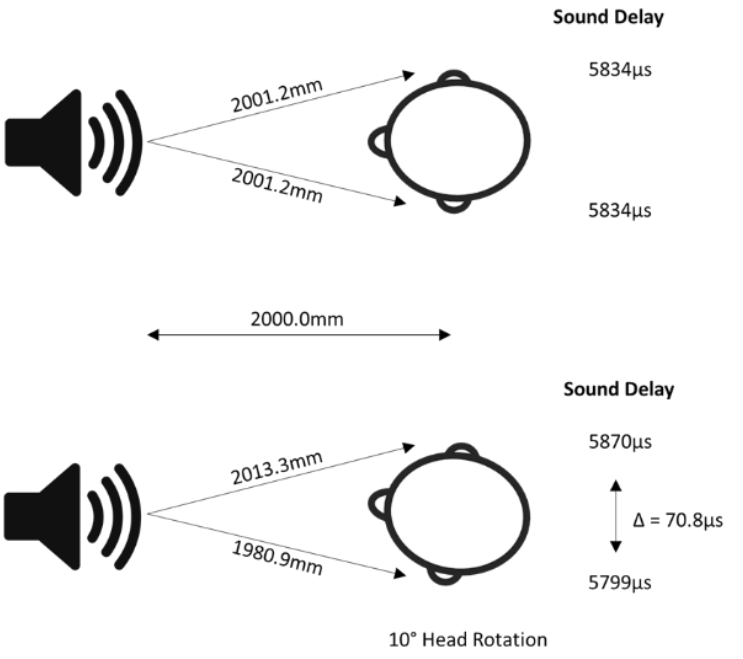


Figure 3.10 Effect of head rotation on sound arrival

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 μs 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 synchronisation 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 synchronisation 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 are probably unaware of each other's existence.)

In most cases, the Acceptors will have received their audio data packets earlier than that common synchronisation 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 drop-outs 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 Synchronisation Point. We'll learn more about the Synchronisation Point in Chapter 4.

However, you can't render the audio at the Synchronisation Point, as it's still encoded. Between the Synchronisation Point and the final rendering point the data needs to be decoded, and any additional audio processing, such as Packet Loss Concealment[20] (PLC), active noise cancellation (ANC), or hearing aid audio adjustments performed, before it can finally be rendered.

---

[20] 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.

The time needed for that 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 an Initiator defined value which specifies the time in microseconds after the Synchronisation Point where the audio is to be rendered in every Acceptor. This is illustrated in Figure 3.11. The "C>P" suffix for the SDU Synchronisation 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.

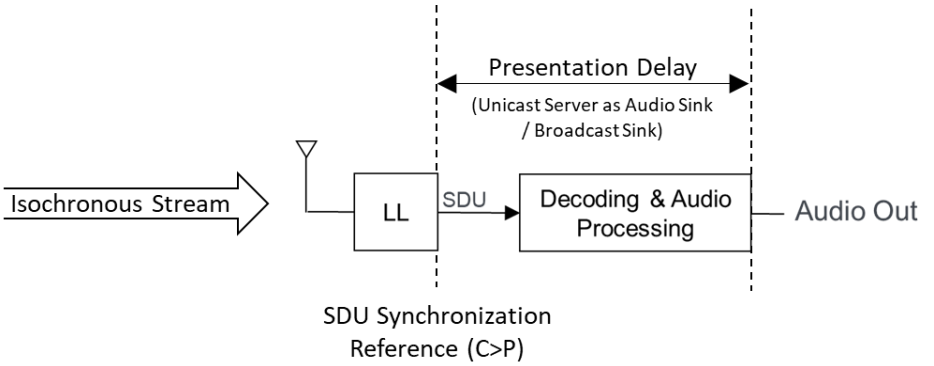


Figure 3.11 Presentation Delay for rendering on an Acceptor

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 synchronise 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 sets 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 and 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 every Acceptor must support) should be avoided as they may introduce echo if the ambient sound is also present, unless they are being used specifically to accommodate this in very large venues. The specifications do not define what a Broadcast Sink should do if the Presentation Delay falls outside the range it can support.

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 specs 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 Stream could be transmitted, as shown in Figure 3.12.

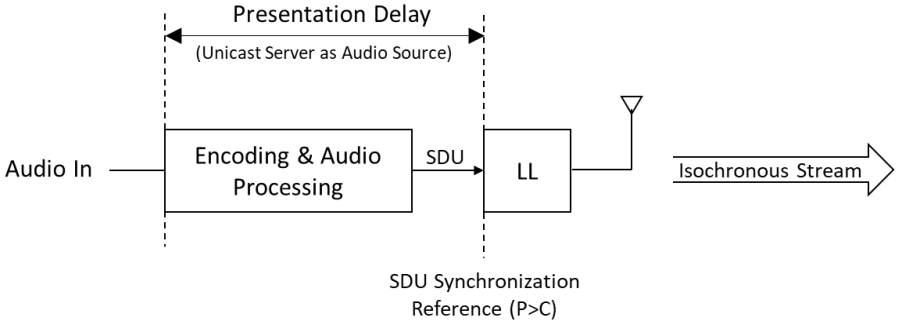


Figure 3.12 Presentation Delay for audio capture

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 3.7 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 incoming streams if it wants to restore that information.

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.

3.11 Announcements

As part of the philosophy of giving more autonomy to an Acceptor, the Bluetooth LE Audio Specifications allow them to transmit Announcements, informing Initiators that they are available to receive or transmit audio data, by using a Service Data AD Type[21]. 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. The LTV structure used in the AD Type field of an Announcement 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	ASCS UUID[22]	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 3.3 AD Values for Targeted and General Announcements

The Common Audio Profile (CAP) describes how Initiators or Commanders (see Section 3.12 below) react to Announcements by defining two specific modes, which helps explain when to use each. These are:

- INAP - the Immediate Need for Audio related Peripheral mode (INAP), and
- RAP – the Ready for Audio related Peripheral mode (RAP).

INAP refers to the case where an Initiator or Commander wants to make a connection, usually due to a user action, and needs to determine which Acceptors are available to connect to. If it discovers an available Acceptor, it should connect, or, if it has discovered more than one, present the user with a range of available Acceptors to let them make that choice. The Initiator or Commander would normally commence scanning at a higher rate to find Acceptors when

[21] AD Data Types are structures used in Bluetooth advertisements to provide information about a device or its capabilities. They are defined the Core Specification Supplement (CSS).
[22] 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.

operating in the INAP mode.

In contrast, when in RAP mode, the Initiator scans at a lower rate, but will respond to a Targeted Announcement from an Acceptor by connecting.

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 it, and are called BAP Announcements [BAP 3.5.3], which contains an Available Audio Context field. If they are used for control purposes or Scan Delegation, they do not. They are then called CAP Announcements [CAP 8.1.1]

Broadcasters use Announcements within their Extended Advertisements to inform Broadcast receivers and Broadcast Assistants (see Section 3.12) that they have a broadcast stream available. These take two forms:

3.11.1 Broadcast Audio Announcements

Broadcast Audio Announcements inform any scanning device that a Broadcaster is transmitting a group of one or more broadcast Audio Streams. The LTV structure used for Broadcast Audio Announcements is shown below in Table 3.4.

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 3.4 AD Values for Targeted and General Announcements

3.11.2 Basic Audio Announcement

The confusingly similarly titled Basic Audio Announcement 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.

3.12 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 infra-red 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 where 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.

That is just the beginning of the problem. 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 Bluetooth 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 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 we use Bluetooth audio devices.

-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 GATT[23] procedures that are part of Bluetooth Low Energy.

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 Central[24] device makes a connection to a Peripheral device, and they exchange

[23] The Generic Attribute Profile defines the procedures which are used with the Attribute Protocol in Bluetooth Low Energy.

[24] 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.

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 devices 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 are relatively low audio quality, used mostly 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. 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 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

---

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.

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 Core 5.2 release is a fundamentally new concept within Bluetooth Low Energy. 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.

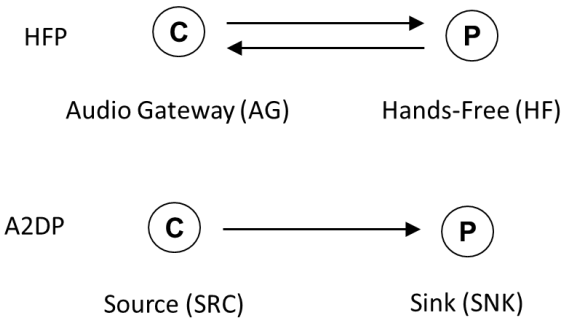


Figure 4.1 Bluetooth Classic Audio topologies

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, which use those Isochronous Streams to send and receive audio data. The asymmetry means that the Peripheral devices can be much lower power. For CISes, they 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, with the Peripheral deciding

which Broadcast Isochronous Streams it wants to receive.

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.

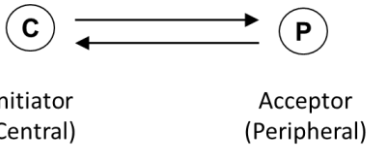


Figure 4.2 Bluetooth® LE Audio roles

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; that task is called the scheduling. The Acceptor is the device that accepts those streams. That concept applies both in unicast and broadcast. An 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.

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 to a left and a right pair of 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.

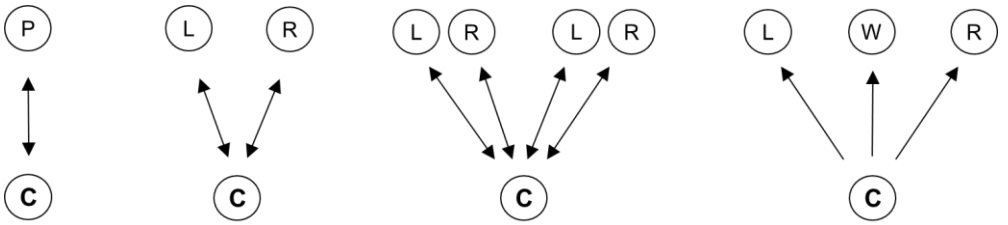


Figure 4.3 Unicast audio topologies

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.

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 Broadcaster 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 audio quality (which typically sets the sampling rate and hence the airtime usage and maximum number of streams), a Broadcaster 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.

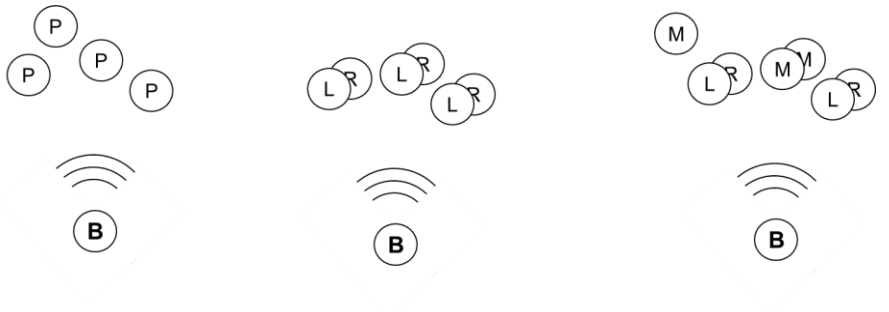


Figure 4.4 Broadcast audio topologies

4.3 Connected Isochronous Streams

To understand 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 or the 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.

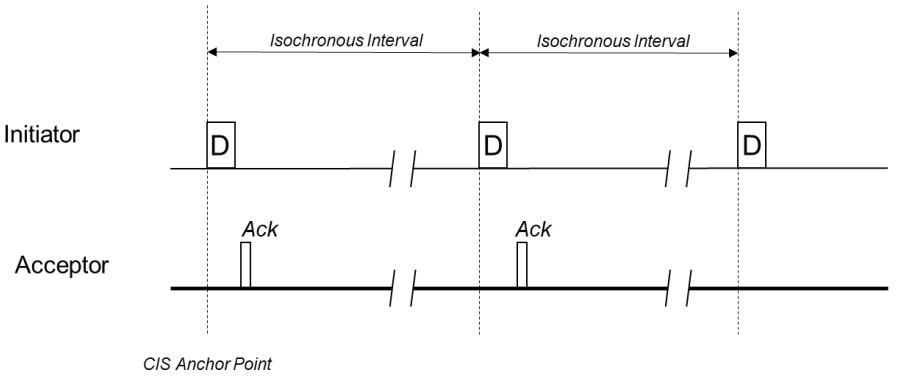


Figure 4.5 Simplified unidirectional *audio transfer*

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 (D) shown in Figure 4.5, which is sent between devices is very simple, and is shown in Figure 4.6.

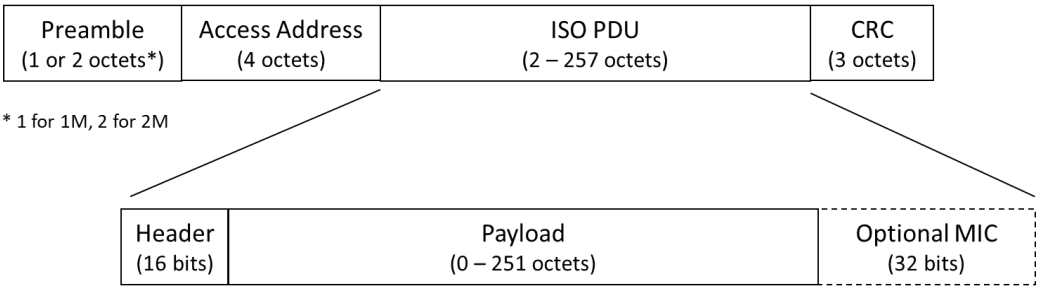


Figure 4.6 Bluetooth® LE Link Layer packet format

The encoded ISO PDU is preceded by a preamble and Access Address, and followed by a CRC. These add 10 or 14 octets when transmitting over an LE 1M PHY[25], (depending on

[25] 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)[26] included with the ISO PDU payload), and 11 or 15 octets when using a LE 2M PHY.

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

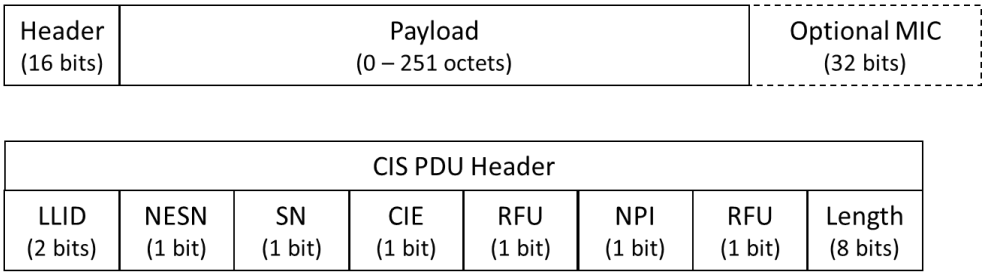


Figure 4.7 ISO PDU format and header for a CIS

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:

- 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 if required, as the CIS structure supports multiple Subevents. Each Subevent starts with the transmission from the Initiator and ends with 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.

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

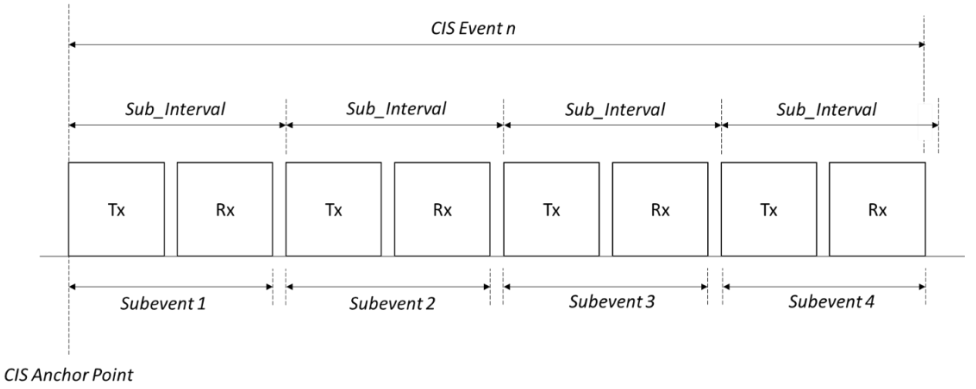


Figure 4.8 Events, Subevents and Sub_Intervals

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 inter-frame spacing, which is defined as being 150µs. 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 transmission channel on every Subevent, as illustrated in Figure 4.9, to protect against interference.

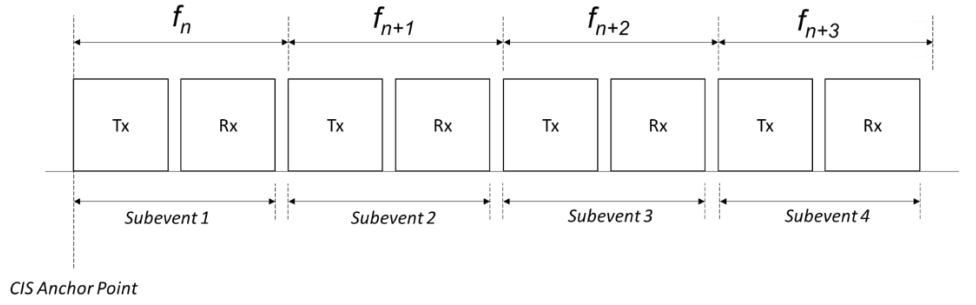


Figure 4.9 Frequency hopping per Subevent

The Core 5.2 specification introduced a new channel selection algorithm, which is more efficient than the one in 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 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.)

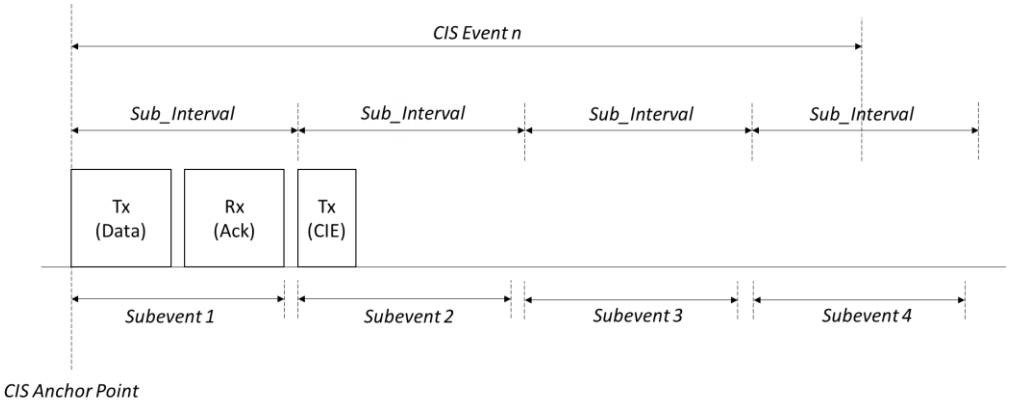


Figure 4.10 Closing a CIS event early with the CIE bit

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 can be useful. For the Acceptor, turning its receiver off until it's ready to do something again can bring a significant power saving.

If the Acceptor does not receive the header with the CIE bit, it will 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, lipsync becomes important.

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 PDUs for a particular CIS.
- The Maximum SDU size for both directions of the CIS
- The SDU interval for both directions

The Maximum Transport Latency affects the overall 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.

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 payloads 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 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 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 transmission (which can have a null PDU payload), 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.

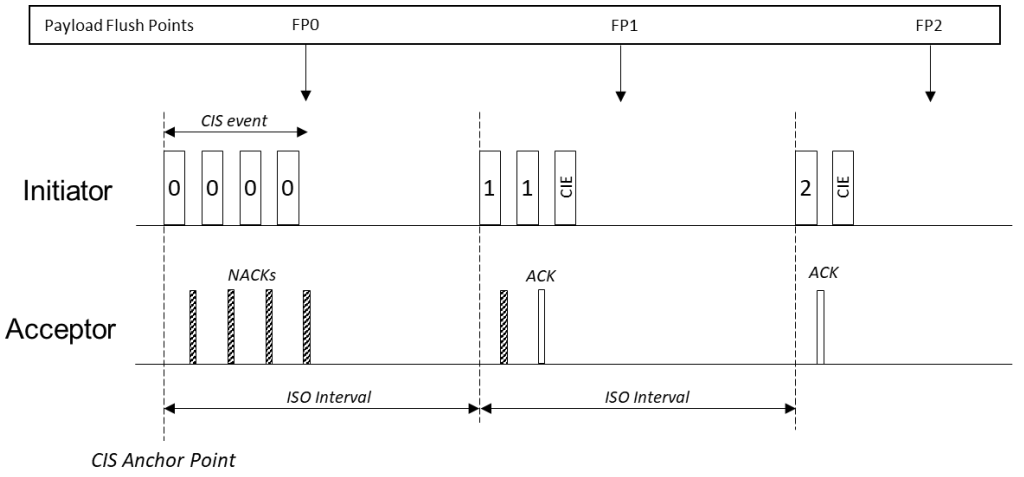


Figure 4.11 A unidirectional CIS with NSE = 4 and FT = 1

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.

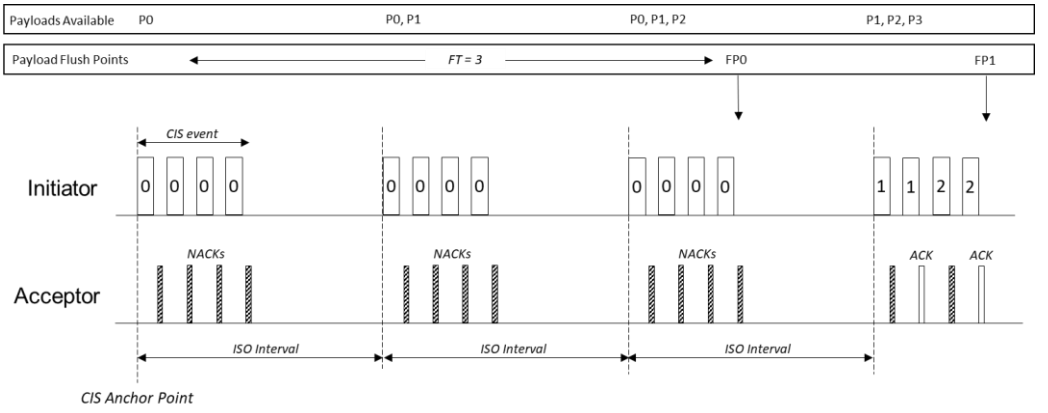


Figure 4.12 An example of FT = 3 and NSE = 4

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 packet has been delivered 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 packets available in each Isochronous Interval. It means that by addressing one problem, we are potentially creating another, which is increasing latency.

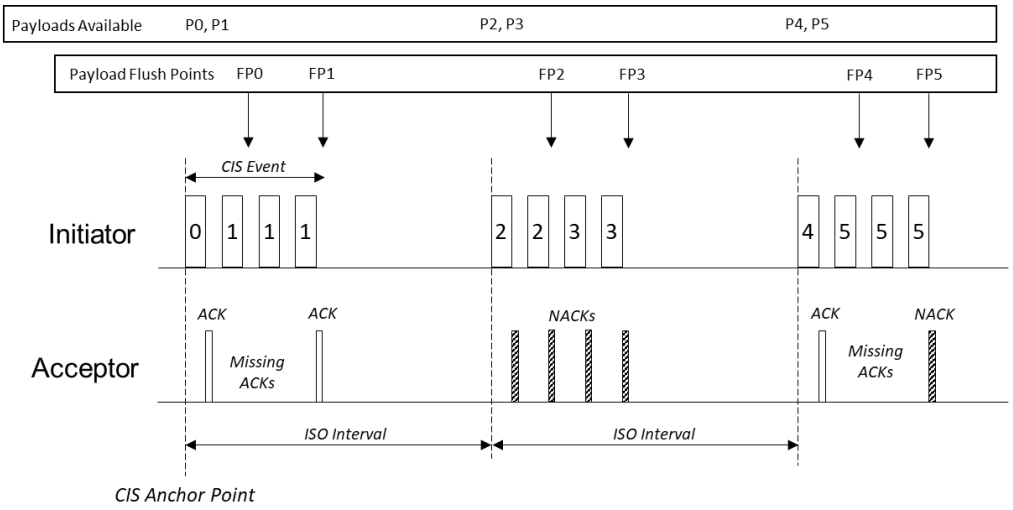


Figure 4.13 The effect of Burst Number = 2, with NSE = 4 and FT = 1

In Figure 4.13, the Isochronous Interval has been doubled, allowing two packets to be supplied in each interval. The combination of the 10ms codec frame and 20ms Isochronous Interval means the Initiator has two PDUs 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 8, 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.

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.

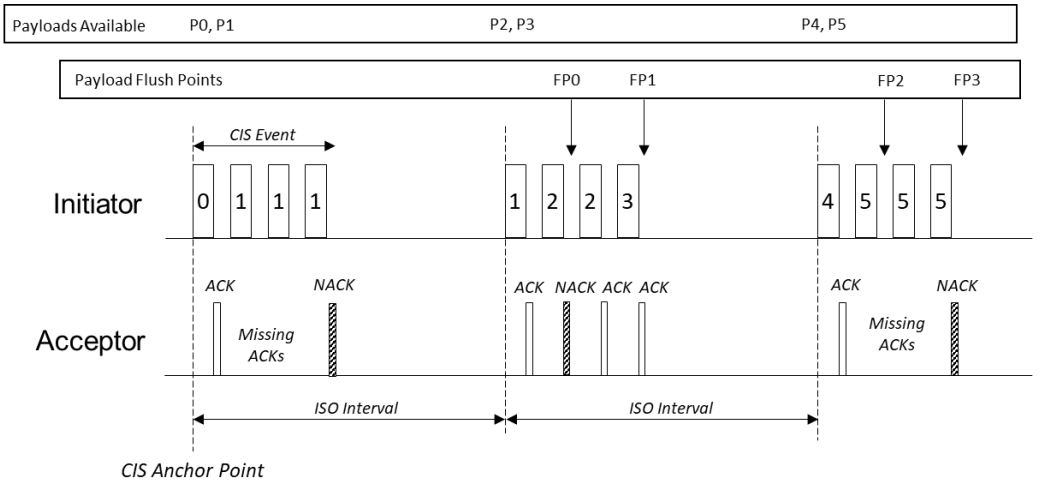


Figure 4.14 Example of NSE=4, BN=2 and FT=2

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 confined to multiple payloads arriving in an Isochronous Interval. It can't be sued to solve the problem of a single payload hogging transmission opportunities which we saw in Figure 4.12.

These parameters cannot by set directly by the Host. It is limited to setting values for the Maximum Transport Latency, the Maximum SDU size and the SDU interval BN, NSE and FT are then calculated in the Controller, which takes into account any other radio requirements in the chip. It is, however, very useful to have an understanding of the potential effects these parameters have on the Isochronous Channel structure. Table 3.19 of TMAP 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 the algorithms in the scheduler, which will be set by the chip supplier.

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 indicates 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 a feature called ISOAL, which is the Isochronous Adaptation Layer, which has been designed to cope with this mismatch. (The LLID bits also indicate when there is no ISO PDU, which is different from the NPI in the ISO PDU header.)

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 audio 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.

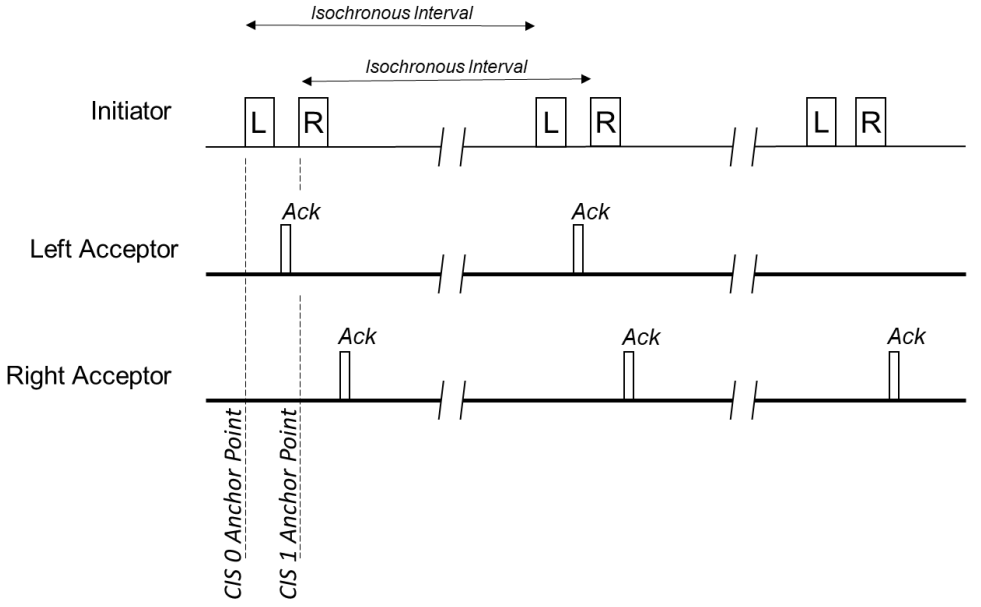


Figure 4.15 Timings for CISes to two Acceptors

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. 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, any associated CISes are terminated. The application then needs to decide what to do with any remaining CISes established with other Acceptors within that CIG.

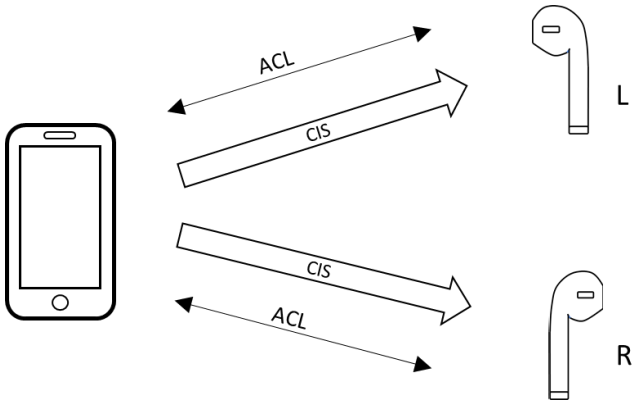


Figure 4.16 Multiple CISes and associated ACL links

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.

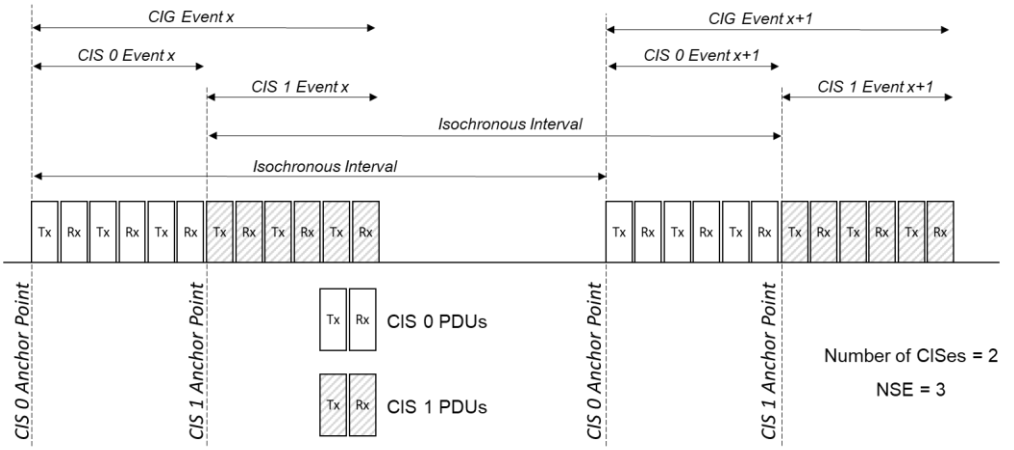


Figure 4.17 Sequential arrangement of two CISes

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.

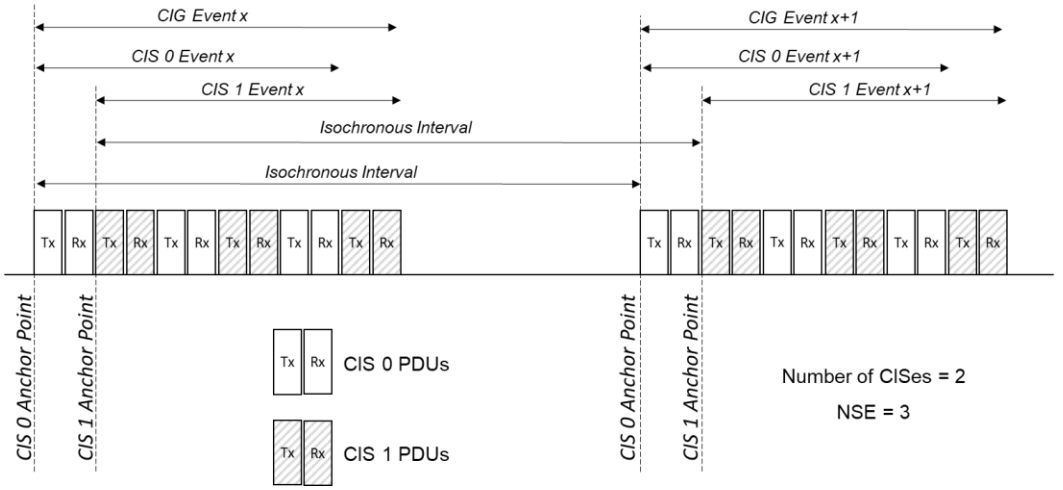


Figure 4.18 Interleaved arrangement of two CISes

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, so that we would use one CIS to transmit 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 acknowledgement packets. It's still a single CIS, but it's now being used for two separate Audio Streams.

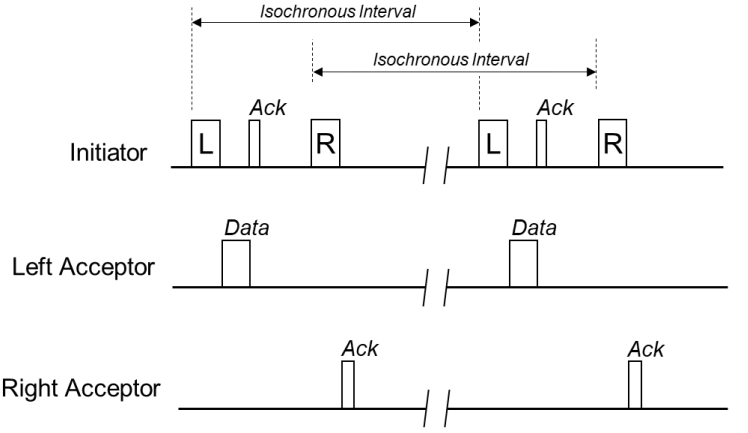


Figure 4.19 Example of a bidirectional CIS for the left earbud and a unidirectional CIS for a right earbud

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 Initiator's "Ack" of the left Acceptor's data would include the CIE bit, occurs before it transmits the data to the right Acceptor shows that it is transmitting the CISes in sequential mode. These "Acks" would also have the CIE bit set.

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.)

Only the Initiator can set the CIE bit in the ISO PDU header in a bidirectional CIS to indicate that they are not ging 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 Acceptor's packet, it should continue to transmit null PDU packets in every available Subevent, to give the Acceptor opportunities to retransmit.

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 Synchronisation 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, 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 Isochronous Channels design includes a synchronisation method which allows microsecond level accuracy of rendering, without an Acceptor needing to have any knowledge of the presence (or otherwise) of any other Acceptors. 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.

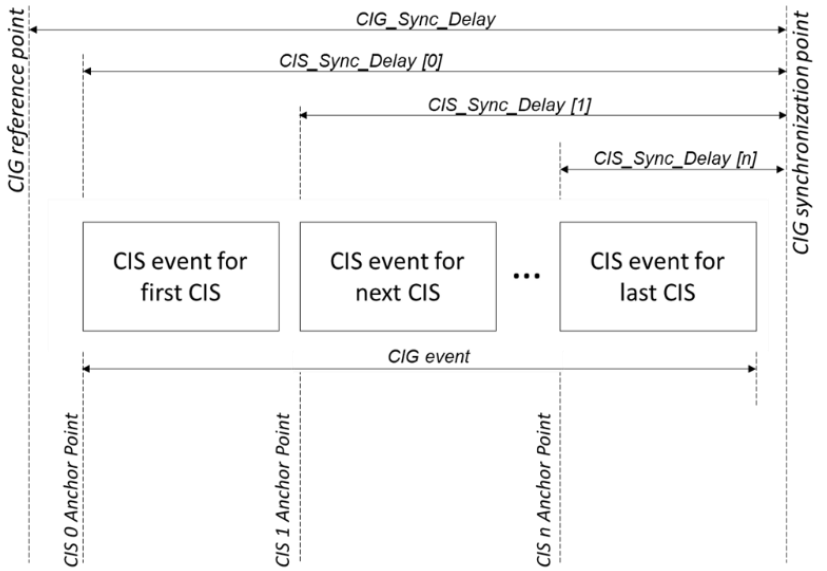


Figure 4.20 Synchronisation of multiple CISes

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 synchronisation 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 Instant[27] for the ACL link associated with that CIS. This allows it to calculate the CIG synchronisation 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 synchronisation 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

[27] An Instant is a timing reference, normally the start of a transmitted packet, which is used for synchronisation. Its use is defined in the Core Vol 6, Part B, Sect 2.4.2.29.

audio data from the Initiator, as it needs extra time to capture and encode the audio stream.

The synchronisation 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, although the method of doing that is down to the implementation. However, the specification means that left and right earbuds can present the two audio streams to the ears around an order of magnitude 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.

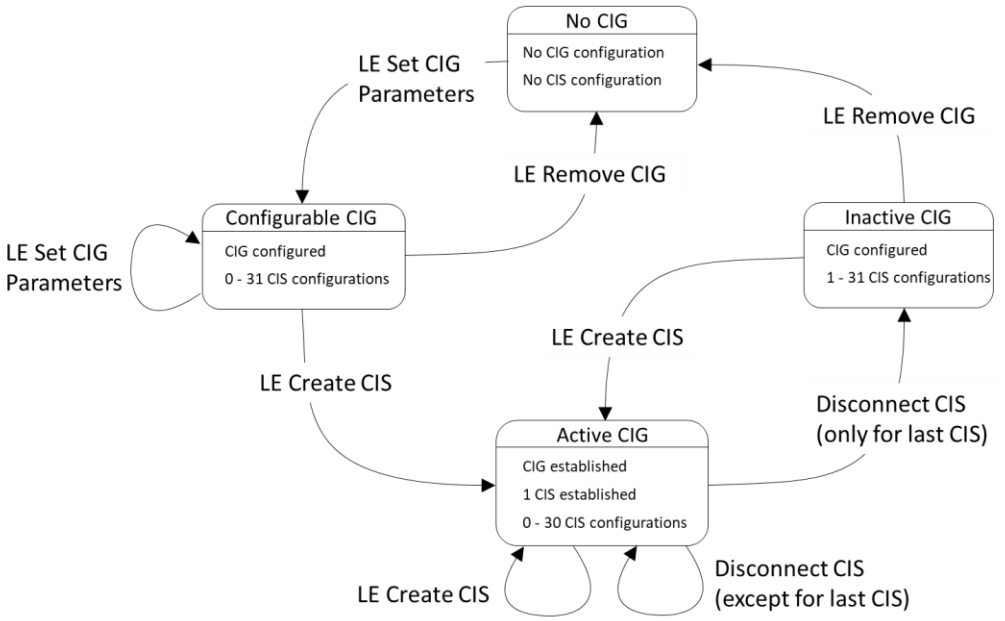


Figure 4.21 The CIG state machine

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 commands[28] called the LE Set CIG

[28] HCI (Host Controller Interface) commands are defined to provide the link between the Host stack

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 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 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 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.

and, if successful, will confirm this with an HCI Complete Command. Note that two of the parameters sent by the Host – 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 are part of a scheduler's choice.

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 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. 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.

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 [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 [7.8.101], leading to both Initiator and Acceptor Hosts receiving an HCI

LE CIS Established event [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 connection[29]. Any device which is within range of a broadcast transmitter can pick up and render 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. Those broadcasts can be received by any and every Broadcast Sink which is within range.

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. A broadcast transmitter is often 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. For public coverage, a Broadcaster can cover a considerable area – far more than can be achieved with a connected Isochronous Stream transmission.

[29] 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.

4.4.1 The BIS structure

The definition of the structure of Broadcast Isochronous Streams and Groups is very similar to what we've just looked at with Connected Isochronous Streams.

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 a 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 length of the payload.

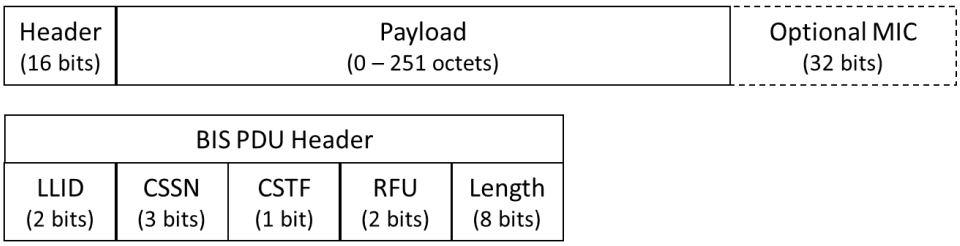


Figure 4.22 Isochronous PDU and header for Broadcast

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. A BIS is composed purely of Subevents which contain data, sent by the Initiator. There is no bidirectional data – everything is transmitted from the Broadcast Source. As before, we see each Subevent is transmitted on a different frequency channel.

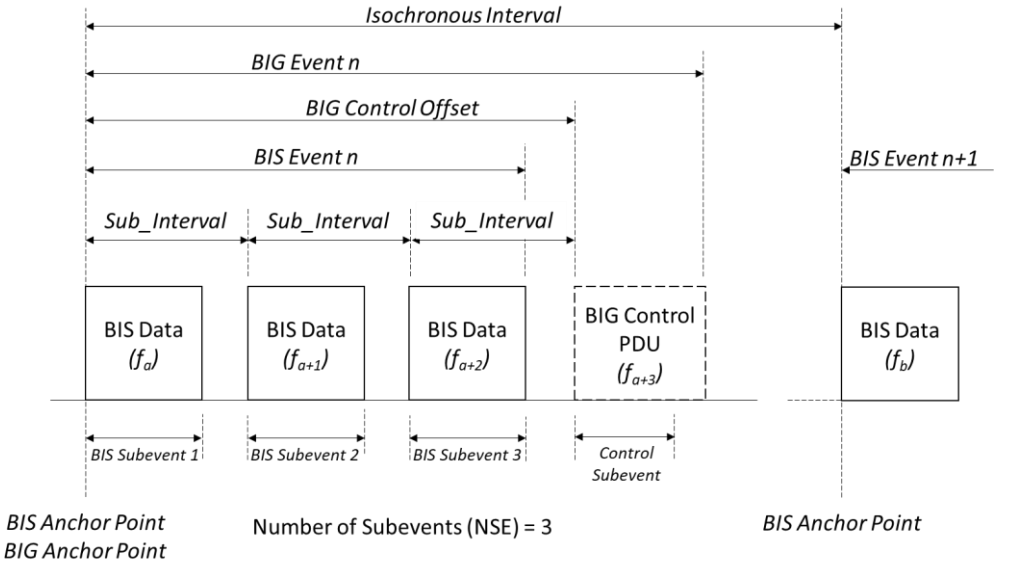


Figure 4.23 Structure of a Broadcast Isochronous Stream

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 in in BIS Events where the header of the ISO PDU 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 that 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.

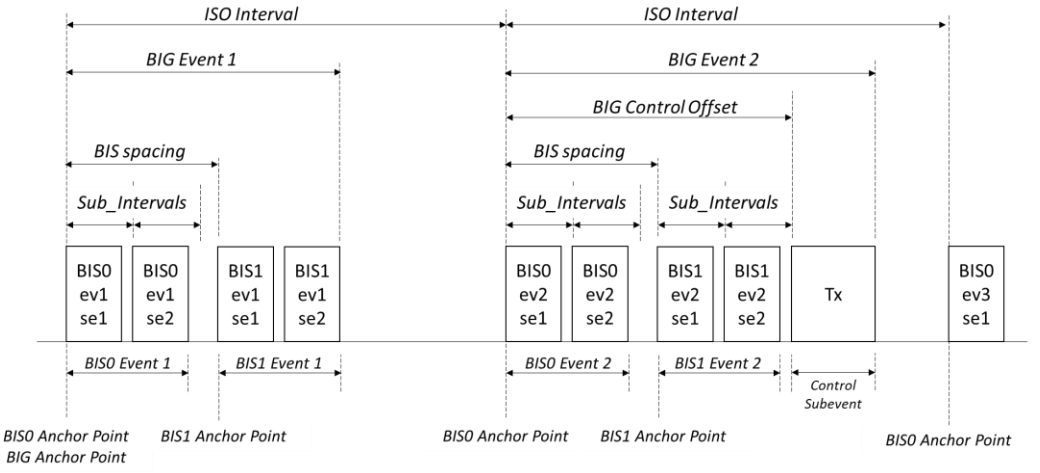


Figure 4.24 Sequential BISes

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.

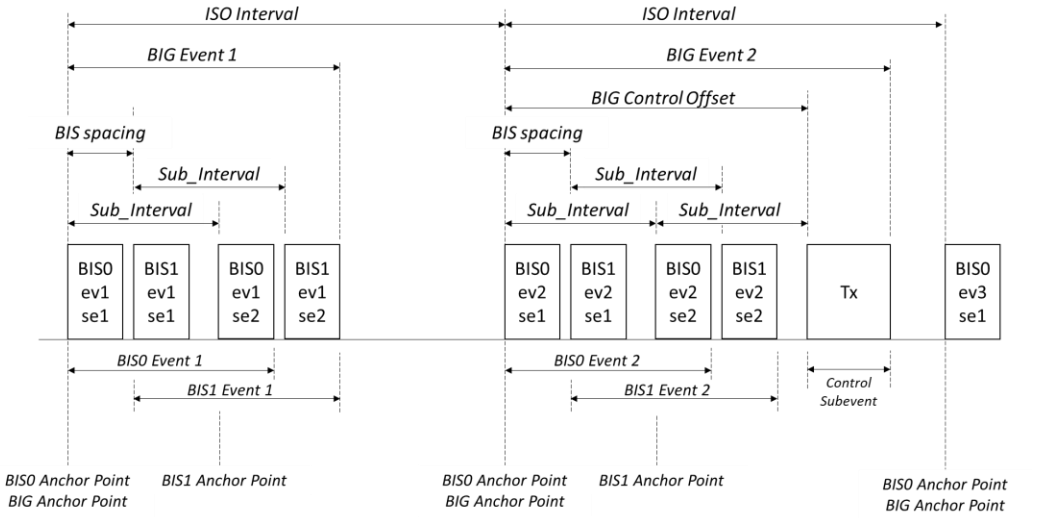


Figure 4.25 Interleaved BISes

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. 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 net 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 transmitted. 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 ones. The downside it 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.

4.4.2 Robustness in a BIS

The way the parameters of a BIS are defined is a little different, 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) 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 within Isochronous Intervals, which is the ratio of NSE to Burst Number (NSE/BN). Group Count 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.)

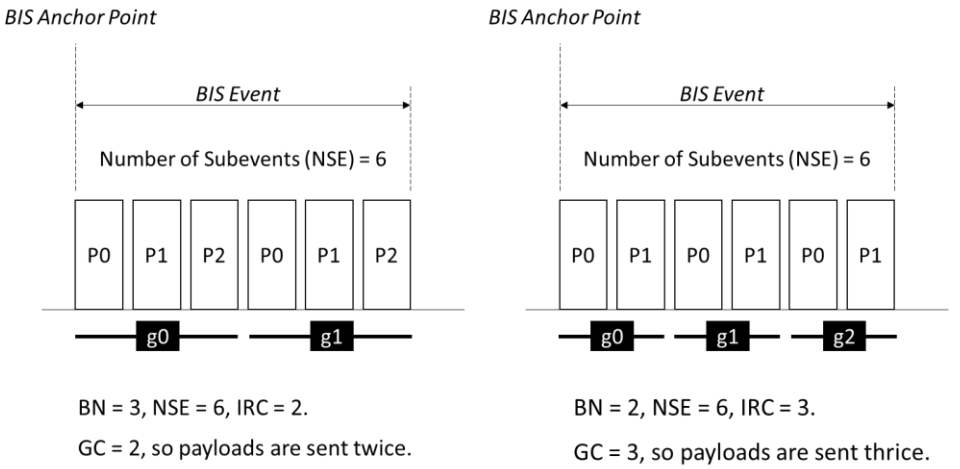


Figure 4.26 The effect of Group Count

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

An important thing to note is that although these two examples look the same, they will have different Isochronous Intervals. Assuming that our frames are 10ms, the example on the left will have an Isochronous Interval of 20ms, as it contains two payloads – P0 and P1. The example on the right will have an Isochronous Interval of 30ms, as there are three payloads, P0, P1 and P2.

Group Count works with two other, new parameters:

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

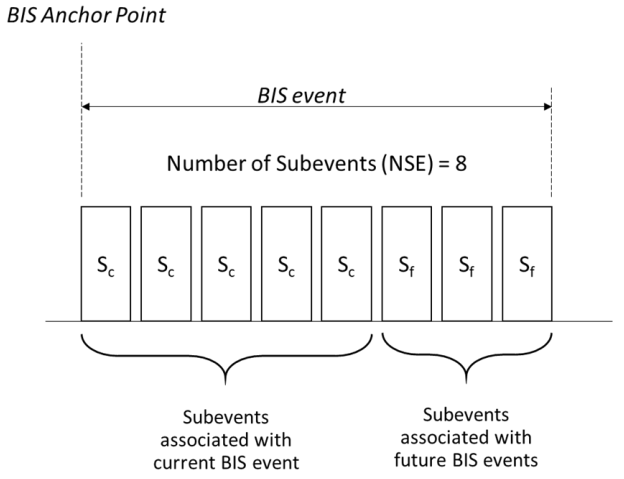


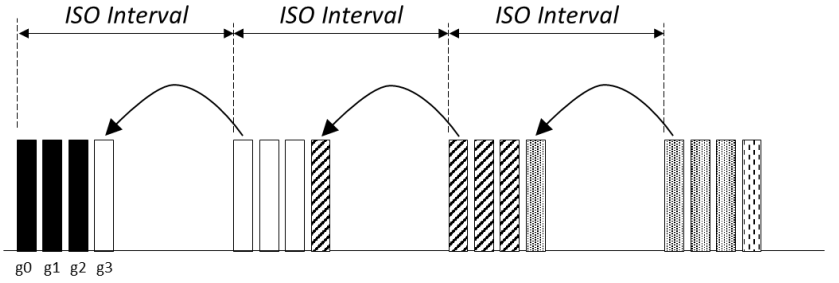
Figure 4.27 The concept of future BIS events

We can now put everything together with some more examples.

These group numbers (g), which we first saw in Figure 4.26, along with the Immediate Repetition Count (IRC), determine which payloads are sent in each transmission slot according to the following rules:

- If g < IRC, group g shall contain the data associated with the current BIS event.
- If g ≥ IRC, group g shall contain the data associated with the future BIS event that is PTO × (g - 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.



	Event x	Event x+1	Event x+2	Event x+3
BIS PDUs	p0,p0,p0,p1	p1,p1,p1,p2	p2,p2,p2,p3	p3,p3,p3,p4

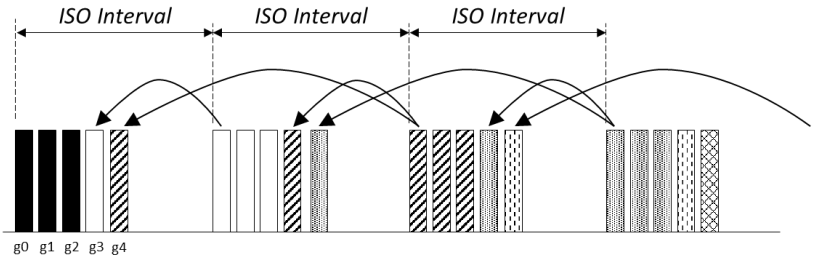
NSE = 4, BN = 1, IRC = 3, PTO = 1. (GC = 4)

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. The conditions shown above show that the last transmission in that BIS event comes from the next BIS event. With this scheme, 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 refence point cannot occur until after the last possible transmission of data, which for p0 is in Event x+1.



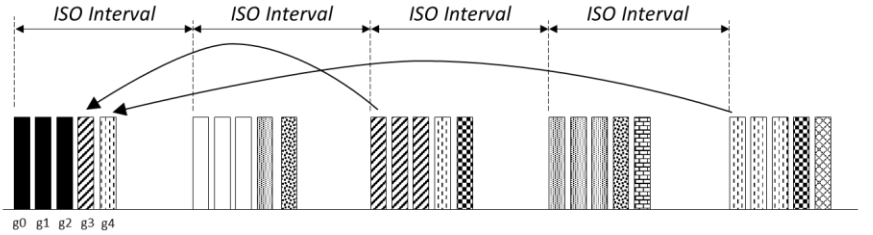
	Event x	Event x+1	Event x+2	Event x+3
BIS PDUs	p0,p0,p0,p1,p2	p1,p1,p1,p2,p3	p2,p2,p2,p3,p4	p3,p3,p3,p4,5

NSE = 5, BN = 1, IRC = 3, PTO = 1. GC = 5

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.



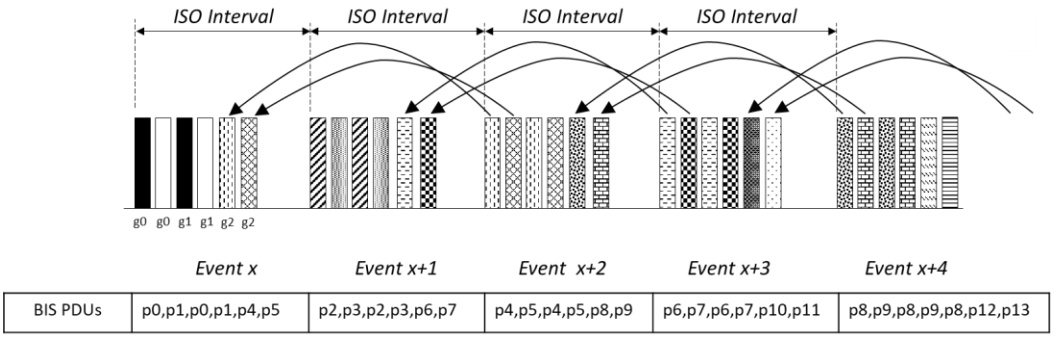
	Event x	Event x+1	Event x+2	Event x+3	Event x+4
BIS PDUs	p0,p0,p0,p2,p4	p1,p1,p1,p3,p5	p2,p2,p2,p4,p6	p3,p3,p3,p5,p7	p4,p4,p4,p6,p8

NSE = 5, BN = 1, IRC = 3, PTO = 2. (GC = 5)

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

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 in order to cope with different requirements in terms of latency and robustness. In the next chapter we'll see how they can be used for different broadcast situations.

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.

4.4.3 The Control Subevent

The Control Subevent [Core, Vol 6, B, 4.4.6.7] 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 synchronised with a BIG and the BISes within it. Without them, a Broadcast Sink would need to continue to receive the Periodic Advertising train and periodically examine the BIGInfo to discover any changes.

A Control Subevent 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. 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. 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.

4.4.4 BIG synchronisation

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 in synchronisation they need to use information that's included within the BIG to understand when they need to render it. To enable them to do this, a BIG Synchronisation 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 Synchronisation 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 Synchronisation 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.

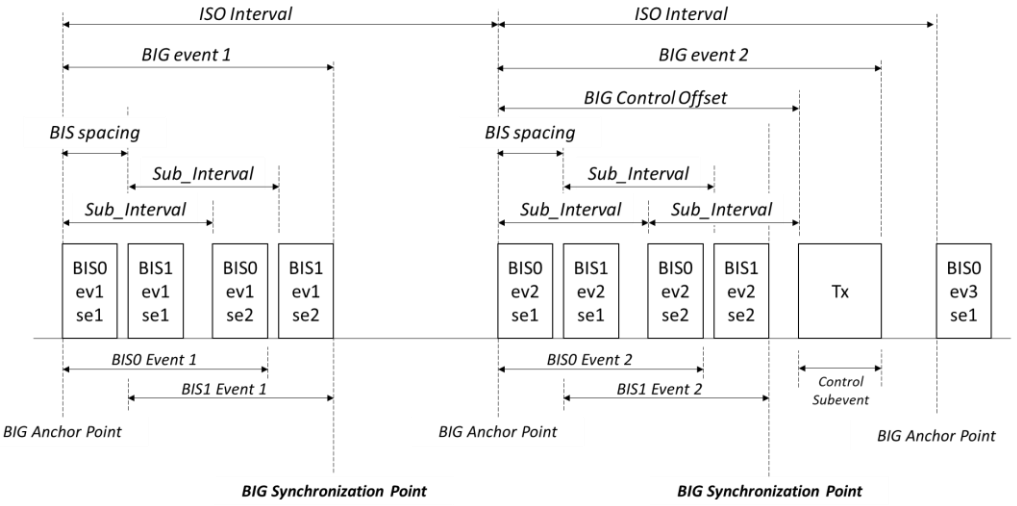


Figure 4.32 BIG synchronisation

At the BIG Synchronisation 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 Synchronisation Point is a fixed point in time at which they can 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 Synchronisation Point – it's rendered at the end of the Presentation Delay, which commences at the BIG Synchronisation Point. As broadcasting consists only of transmissions from a Broadcast Source, there is no concept of Presentation Delay applied to captured data coming back from an Acceptor.

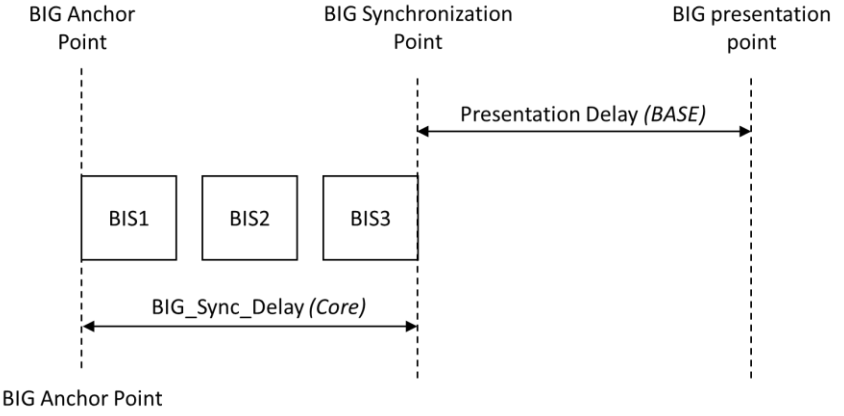


Figure 4.33 The application of Presentation Delay in a BIG

The derivation of the BIG Synchronisation 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 and there is not enough space to allow different settings for individual BISes. This means that the BIG Synchronisation Point is a fixed interval from the BIG Anchor

Point, both of which are known by every Acceptor, as they're 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).

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 a BIS data PDU. 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 much simpler than a CIS, largely because there is no communication with any of the receiving devices, so only two HCI commands are required. LE_Create_BIG configures and creates a BIG, with a Num_BIS number of BISes within it, and the LE_Terminate_BIG command ends and removes it. There are no options to add or remove BISes – everything is done in a single operation. Both commands apply only to an Initiator which is transmitting one or more BIGs.

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 Synchronising 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 Broadcaster 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 synchronise to it.

The method for doing this is called Extended Advertising and was added to Core 5.1. 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.

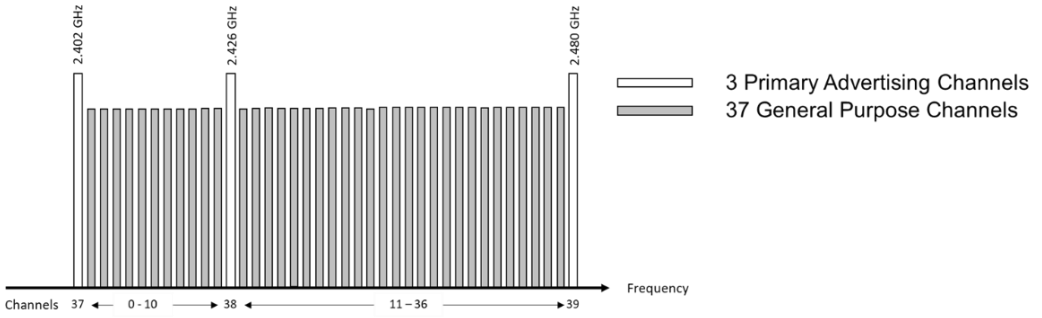


Figure 4.34 Bluetooth® LE channels

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.

Broadcasters 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 within range of an Acceptor, particularly where they're being used in public places, and hence multiple broadcast Isochronous Streams 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 Broadcaster. 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 Core 5.1 allow advertising information to be moved to the general-purpose channels - channels 0 to 36. To accomplish this, a Broadcaster 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. That only becomes apparent when it finds and reads the Extended Advertisement. Figure 4.35 illustrates the basic structure of this scheme.

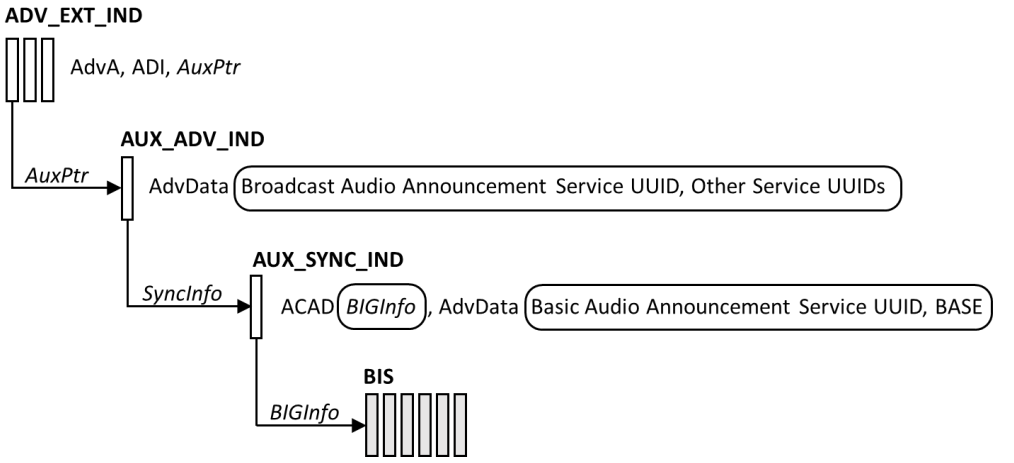


Figure 4.35 Extended advertising for a BIG

The Extended Advertising process is built up using three 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. The AUX_ADV_IND also include a SyncInfo field, which provides information for scanners on how to locate the:
- AUX_SYNC_IND: Auxiliary Synchronisation Information. 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 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. 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. The Broadcaster can even turn off its Extended Advertisements, but keep the Periodic Advertising train running.

The presence of this Periodic Advertising train is indicated by a Syncinfo field in the AUX_ADV_IND PDU of the Extended Advertisement, which provides information on where to find it. Should the amount of information for the Periodic Advertising train be too large to fit into a single PDU, the AUX_ADV_IND can also use an AdvPtr to point to an Auxiliary AUX_CHAIN_IND PDU. In general, the AUX_CHAIN_IND PDUs are not needed.

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 synchronise 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 synchronise 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 and its corresponding 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.

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 4.36 The BIGInfo structure

Figure 4.36 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 CISes 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.

4.4.7 Synchronising 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). An Acceptor can receive any number of BISes within a BIG. In Bluetooth LE Audio, this is called synchronising with a BIG or a BIS. Figure 4.37 shows how this information is used.

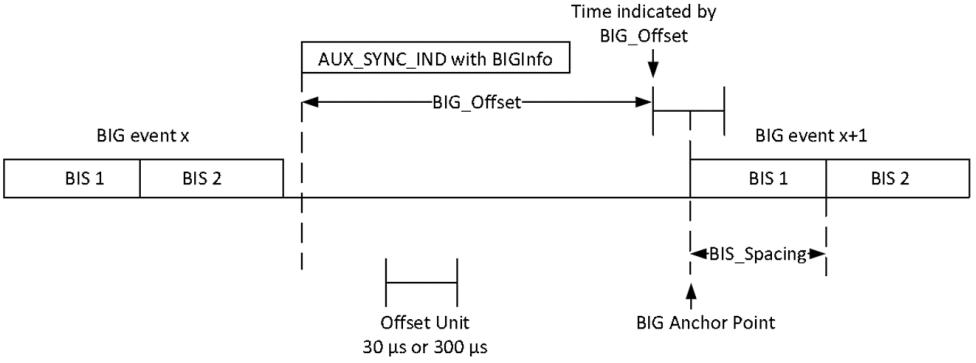


Figure 4.37 Synchronisation timing for a BIG

In most real life situations, a Broadcast Sink would not want to synchronise with all of the BISes. An earbud or hearing aid would normally only want to synchronise to the left or the right Audio Streams of a BIG which included a stereo stream, whereas a pair of headphones would want to synchronise to both 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.

BIGInfo describes the structure of the BIS, telling devices how many BISes it contains and where to find them, but that is just the practical information. It doesn't provide any information about what is in the broadcast Audio Stream. 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 Broadcaster 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 4.38. We will revisit this in more detail when we look at how to set up a Broadcast Stream.

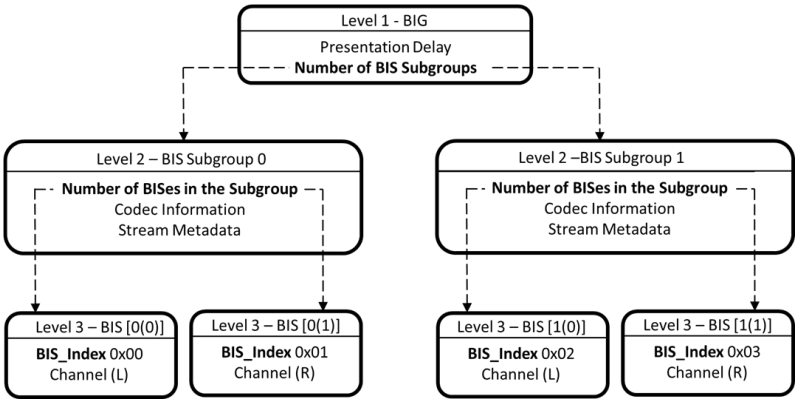


Figure 4.38 Simplified BASE structure

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 ProgramInfo LTV 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 Channel_Allocation, which identifies the Audio Locations it represents, such as Left or Right. Level 3 can include a different set of codec information for specific BISes, which will override the Subgroup value of Level 2. This is only likely to be used where one member of the BIG uses a different setting from the other BISes.

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 synchronise 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 device has all of the information it needs to synchronise 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 4.39).

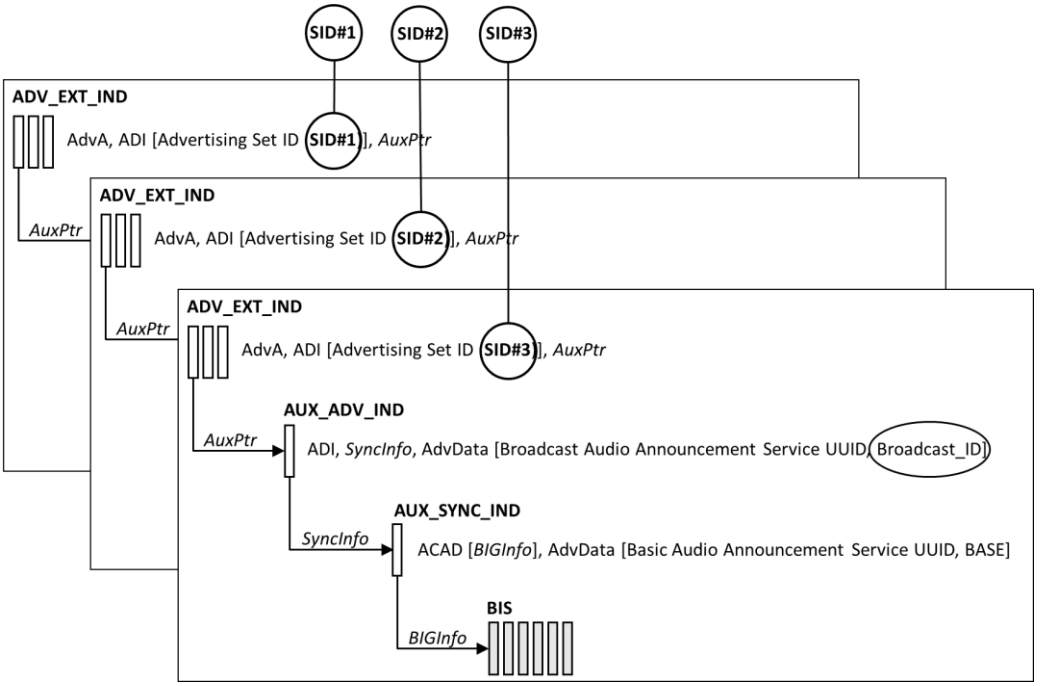


Figure 4.39 Advertising for multiple BIGs

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 the life of the BIG. Broadcast Sinks can take advantage of these IDs to reconnect to a Broadcaster that they know if they recognise the SID and Broadcast_ID.

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. 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 Audio Sharing 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 to receive. A first level of selection can 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 to 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, allows a user to select a broadcast and then instruct a receiving device to synchronise to that 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 Synchronisation Transfer – PAST[30]

A Broadcast Assistant scans for advertisements which indicate the presence of Extended Advertisements in exactly the same way as any other scanning device. Once it discovers them, it can synchronise 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 ProgramInfo.

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 synchronise to the appropriate BISes without having to expend energy in scanning. This process is called Periodic Advertising Synchronization Transfer or PAST.

The level of information provided by a Broadcast Assistant to the user is entirely down to the implementation. A phone app could list every Broadcaster within range; it could limit the

[30] The Core does not use the acronym PAST for Periodic Advertising Synchronisation Transfer, so if you're searching the Core, you need to use the full name. The PAST acronym is introduced in BAP.

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 synchronised 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 Broadcaster or a proxy, or by using an out of band (OOB) method.

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 cope with encryption.

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 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 synchronise 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. 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 Chapter 12

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

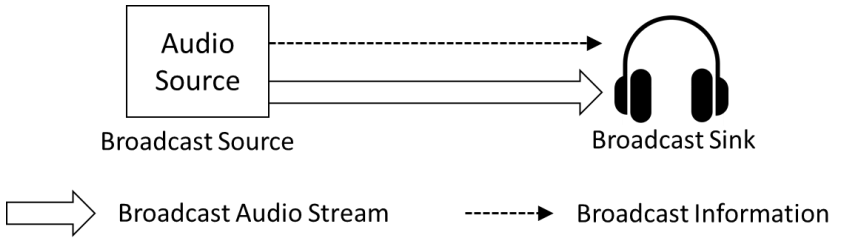


Figure 4.40 Direct synchronisation from headphones

The simplest case, where a pair of headphones does its own scanning to find and select a broadcast audio stream, is shown in Figure 4.40.

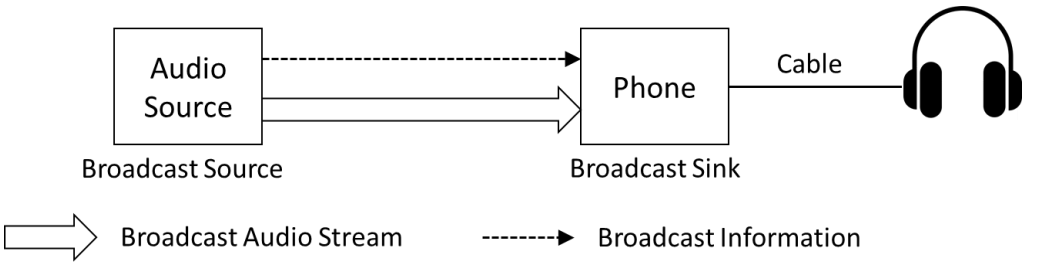


Figure 4.41 Using a phone as a Broadcast Sink

Figure 4.41, is essentially the same, but points out the Broadcast Sink could also be a phone. Here, it can scan for Broadcasters, 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.

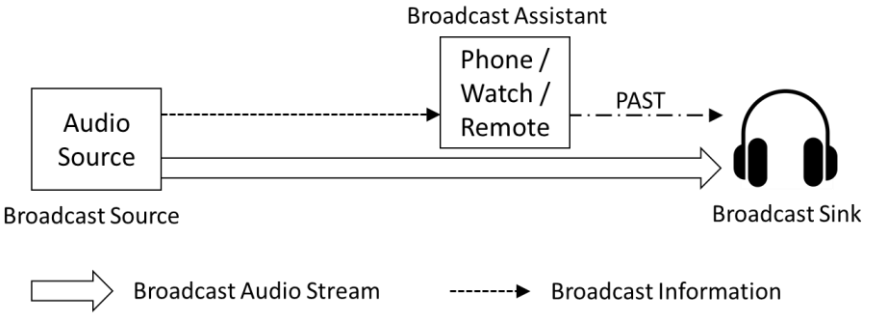


Figure 4.42 Using a Broadcast Assistant and PAST

Figure 4.42 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 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 4.43.

Although there is no connection between a Broadcast Source and a Broadcast Sink for the Audio Stream, devices can use ACL connections to help synchronise to the BIS, so that a TV could automatically synchronise to an Audio Stream when you enter a room. In this case, the Broadcast Source would normally also contain a Broadcast Assistant, which would be paired to a user's headset. When the user chooses to connect to the Broadcast Assistant, it would use PAST to provide details of how to synchronise to the BISes and BASS to transfer the Broadcast_Code.

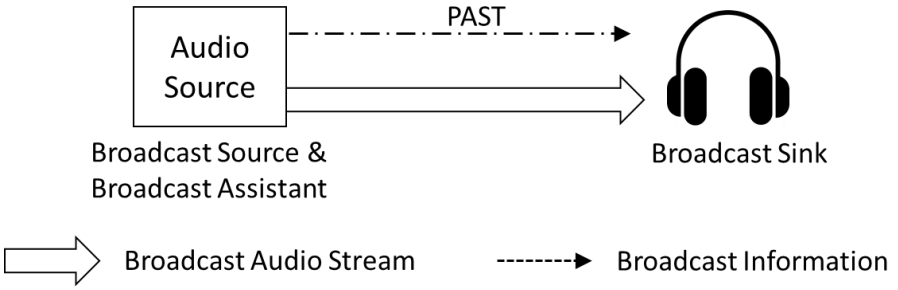


Figure 4.43 Collocation of a Broadcast Source with a Broadcast Assistant

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 frame sizes. The most common reason for this is that an Acceptor only supports a 10ms frame size, but an Initiator needs to use 7.5ms frames, as it is running connections with other Bluetooth Classic devices, such as older mice and keyboards, which are only capable of running at a 7.5ms timing interval[31]. 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 4.44 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.

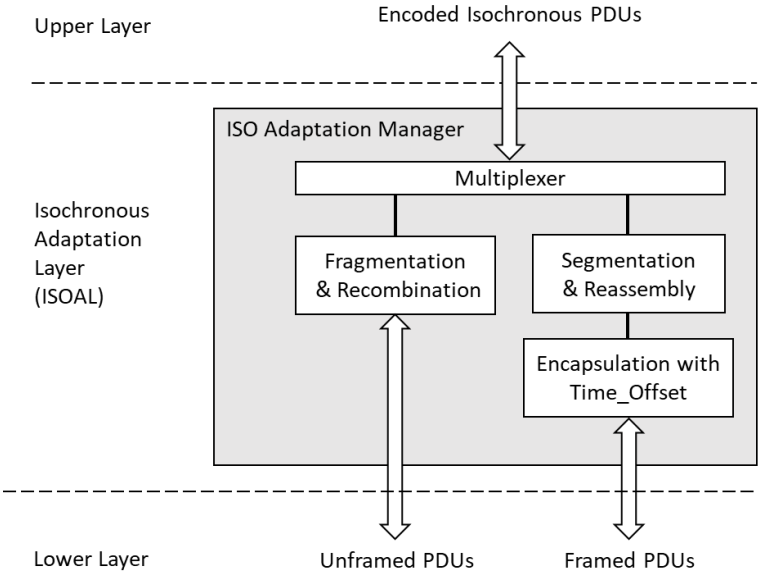


Figure 4.44 Architecture of the Isochronous Adaptation Layer (ISOAL)

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

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 SDUs. The recombination or reassembly processes reassemble them 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 generation of the SDUs need to be synchronised 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 it contains the set of equations which define the Transport_Delay. Transport_Delay is the time between an SDU being presented for transmission and the point where it is ready for decoding at the appropriate Synchronisation Reference.

For framed SDUs, the CIG transport latencies are:

$$\text{Transport\_Latency} = \text{CIG\_Sync\_Delay} + FT \times \text{ISO\_Interval} + \text{SDU\_Interval}$$

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,

$$\text{Transport\_Latency} = \text{BIG\_Sync\_Delay} + (PTO \times (NSE \div BN - IRC)) \times \text{ISO\_Interval} + \text{ISO\_Interval} + \text{SDU\_Interval}$$

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

$$\text{Transport\_Latency} = \text{CIG\_Sync\_Delay} + FT \times \text{ISO\_Interval} - \text{SDU\_Interval}$$

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 QoS choices influence robustness and latency.

Chapter 5. LC3, latency and QoS

5.1 Introduction

Two of the most debated aspect 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 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 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. MP3 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.

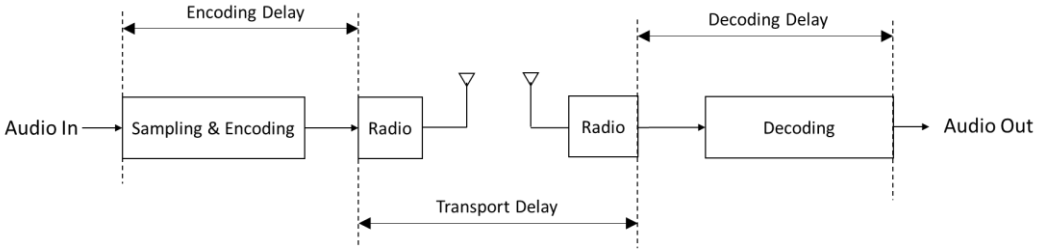


Figure 5.1 The elements of latency in an audio transmission

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.

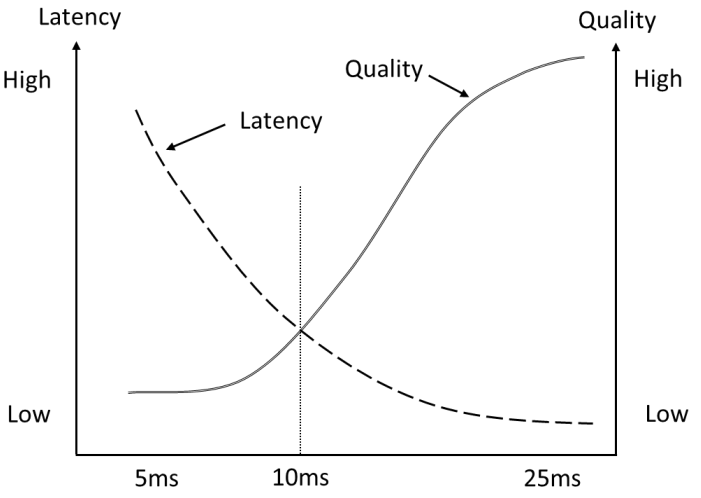


Figure 5.2 The sweet spot for audio codec frame size

Figure 5.2 illustrates the trade-off. This will vary from codec to codec, depending on how they perform the compression, but for a general purpose codec which can be used for both voice and music, the industry has found that there is a sweet spot for the frame length of around 10ms, which gives good quality 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 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 and then converted 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 the design.

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.

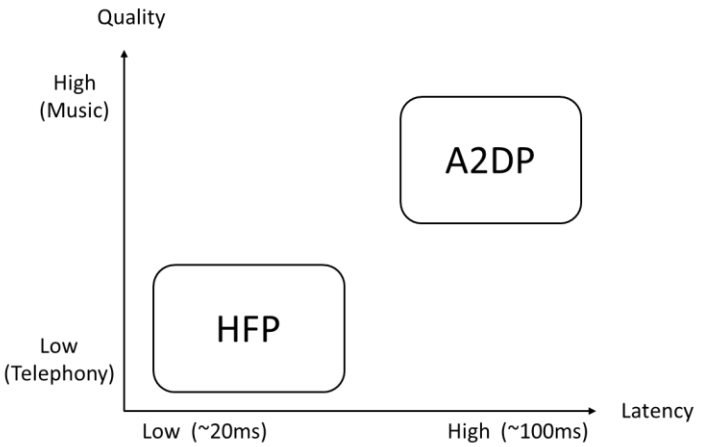


Figure 5.3 Performance of HFP and A2DP profiles

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.

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 alternative[32] codecs which were developed by external companies or standards groups – a selection which includes AAC[33] (which is used by Apple in most of their Bluetooth products), MP3 and ATRAC[34], 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 synchronised. 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 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. Moving to Bluetooth

[32] Referred to as "optional" in older specifications and "additional" in more recent ones.
[33] AAC is the Advanced Audio Codec
[34] ATRAC is the Adaptive Transform Acoustic Coding codec, developed by Sony

requires a codec that is able to cover much more of the Quality / Latency spectrum than SBC.

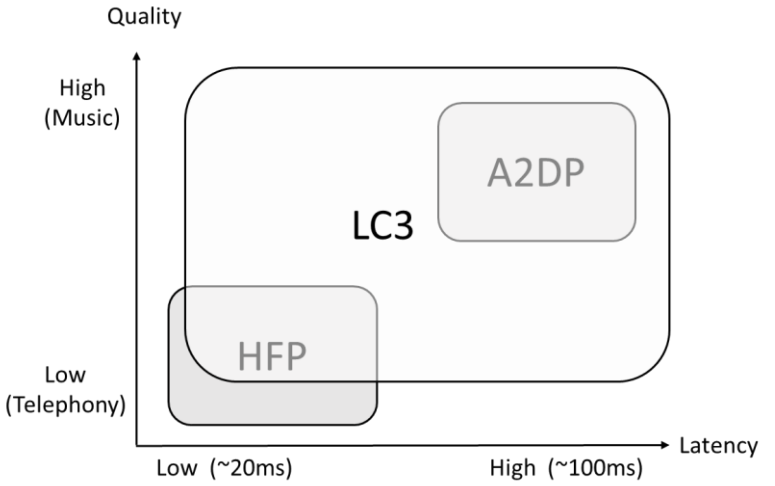


Figure 5.4 LC3 - a more efficient codec

During the Bluetooth LE Audio development, it became apparent that the current Bluetooth codecs would struggle to meet the requirements. Not only were they limited in their quality and latency trade-offs, but SBC is not as efficient as earbud and hearing aid designers would like. It has a relatively low complexity, but takes up too much airtime, which has 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 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 RANDZ[35] 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.

[35] 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. 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 implementation[36].

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). It also 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 (10.88 @ 44.1kHz sampling) 7.5ms (8.163 @ 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	Unlimited by the specification. In practice, 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.

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

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.

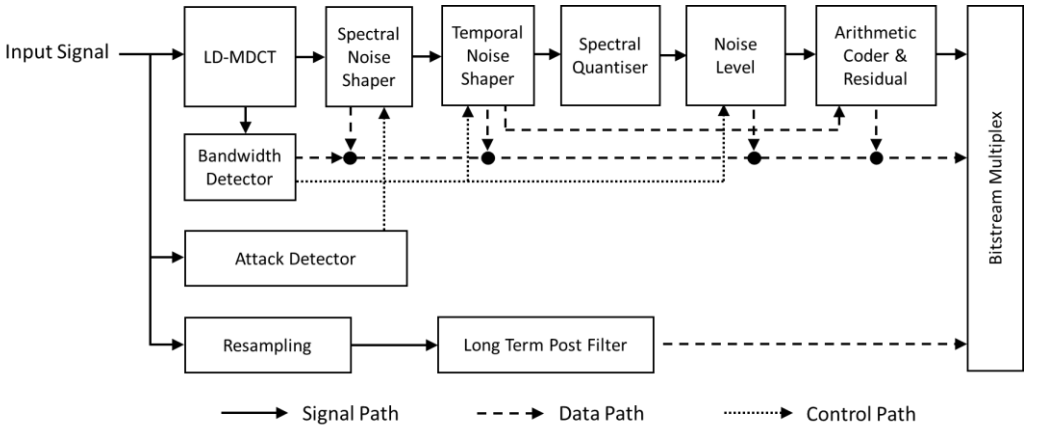


Figure 5.5 High level overview of the LC3 encoder

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 (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 empty 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 (LPTF). 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, which we saw when we looked at the Core.

5.4.2 The LC3 decoder

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

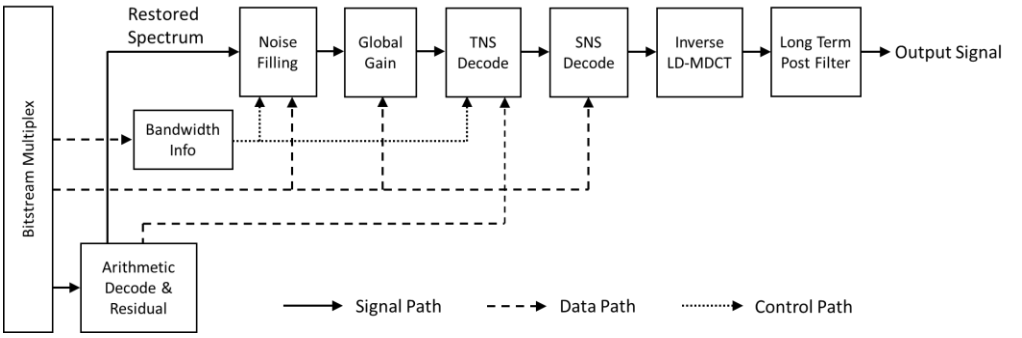


Figure 5.6 Overview of the LC3 decoder

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	Typically 1 or 2, but limited only by the profile or implementation.

Table 5.2 LC3 configuration parameters

The bits per sample at the encoder and decoder are local settings and may be different. In most cases, they are set to 16.

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 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) and 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
441_1	44.1	7.5 ms	97	95.06
441_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.

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 feature of 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 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 it in more detail.

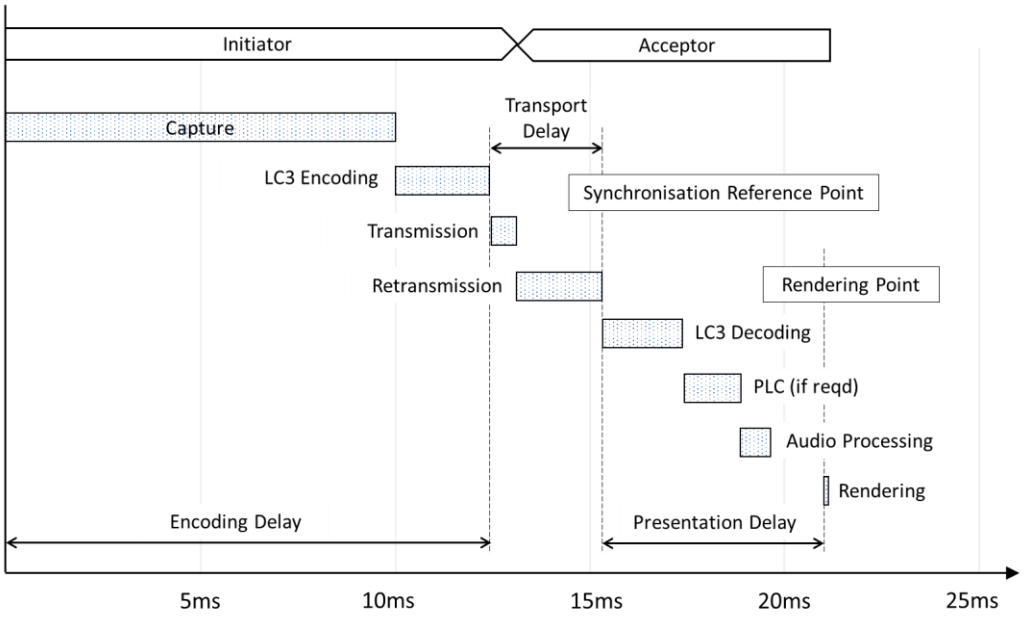


Figure 5.7 The component parts of latency

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 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.

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 Synchronisation 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. The Synchronisation 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 apply the packet loss concealment, if that is required, which takes another few milliseconds. Although it's not needed for most packets, the time to run the 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 over 20ms. 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 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 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 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 allowed, 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.

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.

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
[1] The 44.1 kHz figures differ because they are defined for framed PDUs. All other sampling rates are unframed.						

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 just over 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 actual overall latency values which are likely to achieved. These have been calculated using a value of 12.5ms for the LC3 to sampling and encode a 10ms frame.

	Low Latency			**High Reliability**	
	Unicast and Broadcast			**Unicast**	**Broadcast**
Sampling Frequency	**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

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 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 are recommendations (remember that the Controller uses some of these as guidance, not as definitive values) covering 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. Some higher layer profiles require 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.

Recalling Figure 5.7, the Presentation Delay in that example is only around 5ms, leading to an overall latency below 25ms. Many hearing aids 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. However, a basic Broadcast Source has no way of knowing the capabilities of the Broadcasts 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.

A pair of Acceptors can act unilaterally if they can communicate with each other, by making a decision to render earlier, potentially on the basis of the Context Type for the stream. However, that is outside the scope of the Bluetooth LE Audio specifications.

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.

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 acknowledged or flushed. For Broadcast, it is simply the number of times the PDU should be 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, audio data won't be transmitted RTN times. RTN is better interpreted as the maximum number of retransmissions, not the average number. 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.

	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 Broadcaster 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.

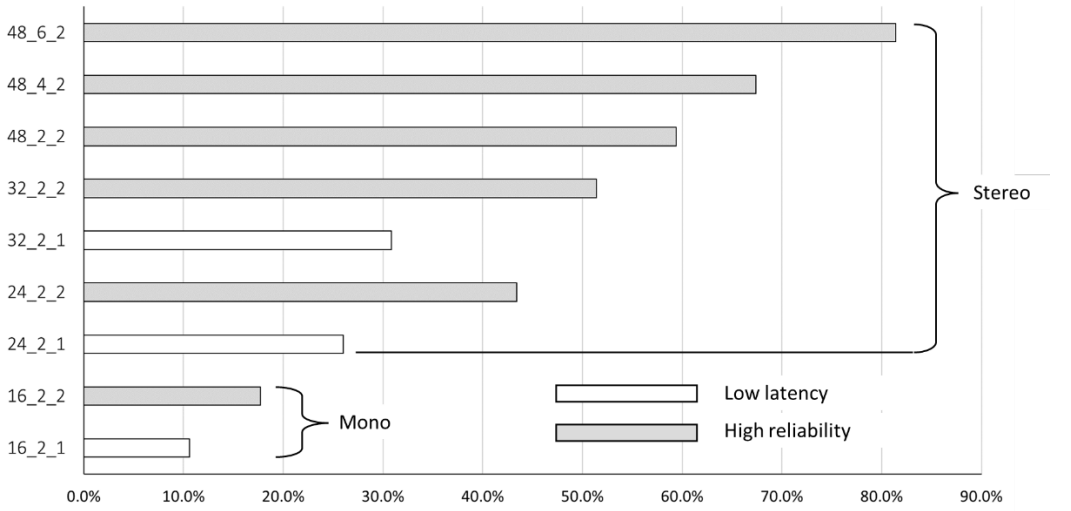


Figure 5.8 Airtime usage for different unidirectional QoS configurations

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.

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 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 and increased output levels. 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 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 website[37].

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 MUSHRA[38] protocol, with a mixture of sounds from the EBU's[39] 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 sampling frequencies in Table 5.9.

---

[37] https://www.bluetooth.com/blog/a-technical-overview-of-lc3/, which includes a link to an audio demo.
[38] Multiple Stimuli with Hidden Reference and Anchor methodology for testing perceived quality, defined in ITU BS.1543-3.
[39] European Broadcasting Union TECH 3253 - Sound Quality Assessment Material recordings for subjective tests

Sampling Rate	Description
8 kHz	Suitable for voice of telephony quality.
16 kHz	Higher quality voice. Adequate for voice recognition applications.
24 kHz	Adequate for music where there are imperfect 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.

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 some QoS choices may exclude the development of new use cases. Some sections of the audio industry will jump onto 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 mean that few, if any listeners will appreciate them. In addition, the resulting latency is 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 32kHzas 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 the 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 a major preference for ease of use.

5.8 Multi-channel LC3 audio

In Chapter 3, we introduced the concept of Audio Channels. 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 many audio channels are multiplexed into a CIS or BIS. The number of arrowheads on each CIS or BIS corresponds to the number of audio channels that it is carrying.

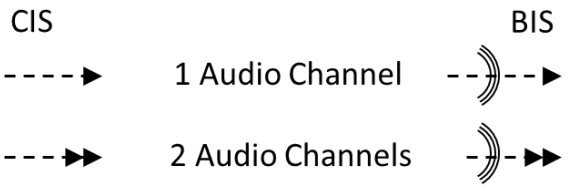


Figure 5.9 Representation of the number of Audio Channels in a CIS or BIS

Figure 5.10 shows the two possible options for transmitting a stereo stream between an Initiator and a single Acceptor. This is the simple unicast use case of streaming music from a phone to headphones or from a TV to a soundbar.

	Channel Allocation	Number of Channels	Sink Audio Locations
(a)	1 ----▶	2	L and R
(b)	2 ---▶▶	1	L and R

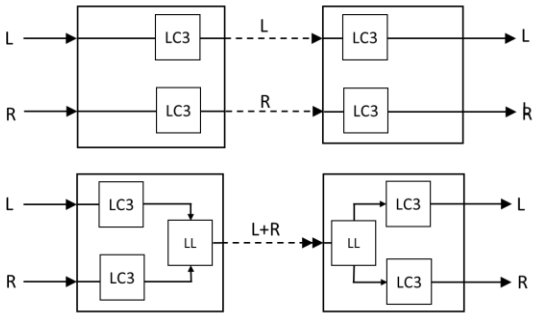


Figure 5.10 Multiplexing options for a stereo stream from an Initiator to one Acceptor

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 concatenated into a single payload by the Controller (the Link Layer – LL), 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 Controller is shown as the LL block (Link Layer) in the diagrams. Where it is not multiplexing a stream, but simply transmitting or receiving a single channel codec frame, it is omitted to simplify the diagrams.)

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 5.11.

	Channel Allocation	Number of Channels	Audio Locations
(a)	1 ----►	1	L and R
(b)	1 ----►	2	L and R
(c)	2 ---►►	1	L and R

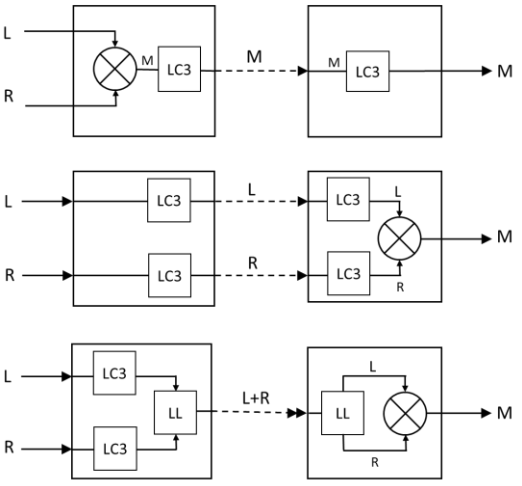


Figure 5.11 Transmitting unicast stereo to a mono Acceptor

For all three options, the Acceptor would set its Audio Sink Locations as Front Left plus Front Right, so its Supported Audio Location would be 0x0000000000000011. Note that there is no mono Audio Location, as mono is a property of a stream, not a physical Audio Location. In Option (a), the Initiator can deduce that the Acceptor will render a mono signal, as it has set the Channel Allocation to 1, which means it will only render the stream to one location. If it had wanted both the left and the right channel, it would have set Channel Allocation to 2. Setting both Audio Location bits, but stating that it only supports a single, non-multiplexed CIS, signifies that it requires the Initiator to downmix the input audio channel to mono before it is sent.

In Option (b) and Option (c), the Channel Allocation and Audio Sink Location information the Initiator receives after the Codec Configuration procedure is identical to what it would have received if the Acceptor were a stereo device (see Figure 5.10). It means that the Initiator has no way of knowing whether it is a mono or a stereo device[40]. 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 this case, the Acceptor is responsible for downmixing the resultant streams.

BAP requires all devices which set multiple Bluetooth LE Audio Locations to be able to support at least the same number of streams, so in Option (a), the Acceptor would also need to be able to support the two CIS configuration of Option (b). If it requires the Initiator to

[40] 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.

perform the downmixing, it would specify the single Channel Allocation and Left plus Right Audio Locations in a preferred PAC record (which we'll cover in Chapter 7), to signal its desire for a single mono stream.

The popular use case of separate earbuds also has different possible configurations. Figure 5.12 shows the two options for transmitting a stereo audio stream to a pair of Acceptors.

	Channel Allocation	Number of Channels	Sink Audio Locations
(a)	1 ----▶	2	L or R
(b)	2 ---▶▶	2	L or R

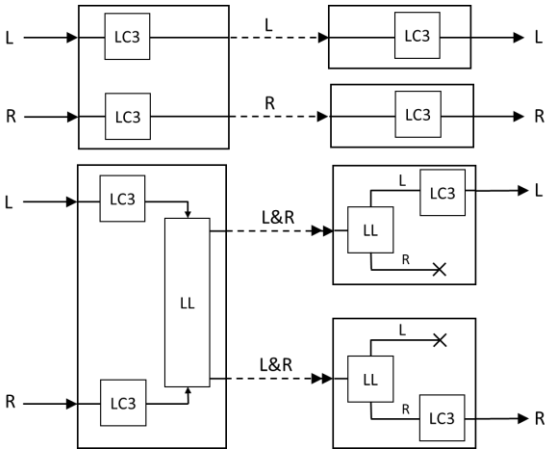


Figure 5.12 Stream options for a pair of stereo earbuds

Option (a) illustrates the configuration which will generally be used, where the earbuds set their Audio Location to just Left or Right (not Left and Right, as in the previous examples). The Initiator will send each earbud a single CIS corresponding to their Audio Location.

In Option (b), both earbuds have set their Channel Allocation to 2, so they will receive a multiplexed stream. Each earbud then has to decode the stream they require, discarding the other. It is a less efficient option, but allows earbuds to swap streams if they detect that the user has turned around, so has application in 3D sound and virtual reality headsets, particularly if additional spatial audio content is available.

The configurations shown above are only a small selection of the possible combinations and not every Initiator and Acceptor will support all of them. BAP lists 14 different combinations of stream configuration (which is not exhaustive), mandating the most common ones, and applying 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 bidirectional streams. 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.

Audio Configurations 6 to 9, and 11 involve two streams. In Table 5.10, they bear the suffix (i), indicating that they refer to the use case where a single Acceptor is involved in both streams and is therefore supporting two CISes.

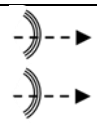
Audio Configuration	Stream Direction (Initiator – Acceptor)	Unicast		Broadcast	
		Initiator	Acceptor	Initiator	Acceptor
1	––––––>	**M**	**M**	Not Applicable	
2	<––––––	**M**	**M**		
3	<––––––>	C	C		
4	––––––>>	O	C		
5	<––––>>	C	C		
6 (i)	––––––> ––––––>	**M**	O		
7 (i)	––––––> <––––––	C	C		
8 (i)	––––––> <––––– >	C	C		
9 (i)	<–––––– <––––––	**M**	C		
10	<<––––––	O	C		
11 (i)	<––––> <––––>	C	C		
12		Not Applicable		**M**	**M**
13				**M**	C
14				O	C

Table 5.10 Stream configuration requirements for single Acceptors, denoted as (i)

Table 5.11 shows the requirements where the Initiator establishes the streams 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	Initiator	Acceptor
6 (ii)	––––> ––––>	**M**	O	Not Applicable	
7 (ii)	––––> <–––––	C	C		
8 (ii)	––––> <––––>	C	C		
9 (ii)	<––––– <–––––	**M**	C		
11 (ii)	<––––> <––––>	C	C		

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

Many other combinations are possible, but a Bluetooth LE Audio device cannot automatically expect them to be supported on an Initiator or Acceptor. An Initiator can determine support for other configurations from the PAC records and Additional_ASE_Parameters values in the Codec_Specific_Configurations.

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.

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. Without that, a connection would default back to the mandatory Bluetooth codec. Normally, additional codecs are licensable from external standards organisations, so anyone can integrate them into their product. Currently, no additional codecs are defined for

Bluetooth LE Audio.

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 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. That requirement 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 both would render audio streams, but only one need 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.

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 means that there must be 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.

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 device[41]. 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.

If you are shipping Acceptors which are part of a Coordinated Set, then CAP demands that 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 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

[41] A firmware update is considered to be the start of a new life.

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 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 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.

Role / Component	Acceptor	Initiator	Commander
BAP Broadcast Assistant	X	O	C
BAP Scan Delegator	C	X	C
VCP Volume Controller	X	-	C
VCP Volume Renderer	O	X	X

Component \ Role	Acceptor	Initiator	Commander
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
CSIP Set Coordinator	X	C	M
CSIP Set Member	C	X	X
"M" = Mandatory, "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

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, as that is set up unilaterally, with no knowledge of whether any Acceptors are present.

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 staring the broadcast stream, 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 ties together volume and rendering control with 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 Connection establishment procedures

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 section 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". These are the "Immediate Need for Audio related Peripheral" mode and the "Ready for Audio related Peripheral" mode.

6.2.7.1 Immediate Need for Audio related Peripheral (INAP) mode

The INAP mode is use when an Initiator 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. As this is generally a high priority response, the Initiator should use the quick setup parameter values for connections from BAP Section 8. The Initiator should 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. If more than one Acceptor responds, the Initiator should present the user with the available options.

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. 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.

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 is 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.

-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 at this stage 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 exposes 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 include information about the codec and the configurations which it supports, 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 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 what 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" eSCO[42] parameter sets for mSBC. The aim of PACS is to prevent those problems arising.

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.

PAC Record	**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 ith PAC Record.

[42] 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.

PAC Record	Description
Codec_Specific_Capabilities [i] (length varies)	Identification of the capabilities of the codec implementation in the ith PAC Record, normally expressed as a bitfield.
Metadata length [i] (1 octet)	Length of the metadata associated with the ith PAC Record. 0x00 if there is none.
Metadata [i]	LTV formatted metadata for the ith PAC record

Table 7.1 Format of a PAC Record

7.1.2 Codec Specific Capabilities

Every Bluetooth LE Audio specification includes 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 described in the Host Controller Interface Assigned Number list, and not the Audio Codec ID defined in the Audio/Video assigned numbers list.

The Codec_Specific_Capabilities requirements for LC3 are defined in BAP Section 4.3.1 and consist of five LTV structures. Each LTV has a Type code, defined in the Generic Audio Assigned Numbers document, so that it can be identified by the device reading it.

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) 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. This LTV is mandatory. (Note that this structure is used for any codec. LC3 only specifies 8, 16, 24, 32, 44.1 and 48 kHz sampling frequencies, shown as lightly shaded).

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 (0x01)

7.1.2.2 The Supported_Frame_Durations LTV

The second codec capabilities LTV structure is the Supported_Frame_Durations (Type = 0x02) 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.

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 (0x02)

7.1.2.3 The Supported_Audio_Channel_Counts LTV

The third LTV structure is the Supported_Audio_Channel_Counts (Type = 0x03) 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 indicates 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.

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 (0x03)

7.1.2.4 The Supported_Octets_Per_Codec_Frame LTV

The fourth LTV structure is the Supported_Octets_Per_Codec_Frame (Type = 0x04) shown in Table 7.5. This structure consists of two pairs of two octets, with the lower pair (Octets 0 and 1) specifying the minimum number of octets per codec frame and the upper pair the maximum number of 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 bitrates[43] supported for the selected sampling rate (see Table 5.1). This LTV is also mandatory.

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 (0x04)

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), which is a single octet stating the maximum number of codec frames which can be packed into a single SDU. It can be omitted

[43] The bitrate is eight times the number of octets.

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 Codec Specific Capabilities LTV structures section of the Generic Audio Assigned Numbers document. 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. It would normally only be present if the Acceptor preferred specific PAC records to be used for specific use cases, 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.

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). If it exposed these as four separate PAC records, they would look like Table 7.6.

PAC Record	**0**	**1**	**2**	**3**
Codec_ID (LC3)	0x0D	0x0D	0x0D	0x0D
Supported_Codec_Specific_Capabilities_Length	0x0B	0x0B	0x0B	0x0B
Supported_Codec_Specific_Capabilities				
Supported_Sampling_Frequencies	*(Example: 16 kHz = 0x04, 24 kHz = 0x10)*			
Length	0x02	0x02	0x02	0x02
Code	0x01	0x01	0x01	0x01
Value	0x04	0x04	0x10	0x10
Supported_Frame_Durations	*(Example – 10ms only)*			
Length	0x02	0x02	0x02	0x02
Code	0x02	0x02	0x02	0x02
Value	0x02	0x02	0x02	0x02
Supported Audio Channel Counts	*(Example: 1 = 0x00; 2 = 0x01)*			
Length	0x02	0x02	0x02	0x02
Code	0x03	0x03	0x03	0x03
Value	0x00	0x01	0x00	0x01
Supported Octets per Codec Frame	*(Example – 40 or 60 = 0x2828 or 0x3C3C*			
Length	0x03	0x03	0x03	0x03
Code	0x04	0x04	0x04	0x04
Value	0x2828	0x2828	0x3C3C	0x3C3C
Supported Max Codec Frames Per SDU	*(Optional. Default = 1)*			
Length	0x02	0x02	0x02	0x02
Code	0x05	0x05	0x05	0x05
Value	0x01	0x01	0x01	0x01

Table 7.6 Example of PAC records for mandatory 16 & 24 kHz sampling at 10ms, with 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, 16_2_2, 24_2_1 and 24_2_2 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 0x00 and 0x01 to support one and two channels.
- The Supported_Octets_per_Codec_Frame has values of 0x2828 and 0x3C3C 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:

PAC Record	**Record 0 (L-T-V)**
Codec_ID (LC3)	0x0D
Codec_Specific_Capabilities_Length	0x0B
Codec_Specific_Capabilities	
Supported Sampling Frequencies	*0x 02 01 14*
Supported Frame Durations	*0x 02 02 02*
Supported Audio Channel Counts	*0x 02 03 03*
Supported Octets per Frame	*0x 03 04 3C28*
Supported Max Codec Frames Per SDU	*0x 02 05 01 (Could be omitted as default = 1)*

Table 7.7 The 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. 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 should 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 the do the same thing for TMAP would require at least ninety-six individual PAC records. As we will see shortly, the ASE 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 is a large number of 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 an associated Sink_Audio_Locations and Source_Audio_Locations characteristic 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 the 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 they are present, at least one bit must always be set. If they're not present, the device should accept any Audio Location proposed by an Initiator. In many cases, the values 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. A speaker is likely to have its Sink_Audio_Locations characteristic set to both Front Left and Front Right. When an Initiator wants to establish a stream, it will decide which Audio Stream to send to match the Audio Location. If it finds two speakers, it would normally send a left stereo stream to the one exposing the Front Left location and a right stereo stream to the one with the Front Right location. If they both say they can support both Front Left and Front Right (which most will do), the user would normally use an application to configure which is which. The Sink and Source Audio Locations characteristics may be writeable, which would allow a configuration utility to set speaker locations for use by other Initiators. Because these are all interoperable features in the Bluetooth LE Audio specification, it means that any compliant audio application should be able to perform this sort of configuration. Alternatively, a speaker manufacturer could provide a physical switch on its speakers to allow a user to specify whether a speaker receives a left or right stream.

In Chapter 5, we looked at how Initiators and Acceptors would deal with downmixing stereo streams for mono reproduction, showing that this could be done either before encoding the Audio Stream, or after reception and decoding. An Acceptor can use its PAC records to indicate to an Initiator whether or not it is capable of downmixing. If it sets its Audio Locations for Front Left and Front Right, but only supports a single Bluetooth LE Audio Channel, then it implies that it is not capable of downmixing, but will only render the stream it has been supplied. If it shows support for two channels, the implication is that it can decode them into separate streams to be rendered. It is then down to the Acceptor implementation to render them as:

- separate left and right Audio Channels, which it would do if it were a sound-bar or stereo speaker,
- a single one of the two Audio Channels it decodes, which it might do if it were an earbud or hearing aid, or
- downmix them to a single mono Audio Channel, if it contained a single speaker.

The latter two options would usually require some level of user configuration of the speaker, but that is down to implementation.

The Source_Audio_Locations characteristic behaves in the same way, although in most cases it is likely to be limited to support of Front Left and Front Right. If the Acceptor supports a single 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. However, 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 specifications.

7.1.5 Supported Audio Contexts

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.

Given this behaviour, an Acceptor should set every Context Type bit in the Supported_Audio_Contexts characteristic where it thinks it will ever have a reason to make that use case unavailable. (An Audio Sink is prohibited from being unavailable to everything, as the Assigned Numbers document prohibits an Available_Audio_Contexts value of 0x0000.)

7.1.6 Available 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. To allow an Initiator to continue with 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.

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.

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's, 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.

	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 for Supported_Audio_Contexts and Available_Audio_Contexts with «Unspecified» unavailable

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:

- Allowing an Initiator to proceed with establishing an Audio Stream,
- Prohibiting it from starting the Audio Stream, or
- Saying it doesn't care – have a go.

The combination of Not Supported and Available is not allowed.

Supported Audio Contexts	Available Audio Contexts	Interpretation for Initiator
0b0	0b0	Audio Stream Context Type may be mapped to «Unspecified»
0b0	0b1	This combination is not allowed
0b1	0b0	An Initiator shall not establish an Audio Stream with this Context Type
0b1	0b1	An Initiator may establish an Audio Stream with this Context Type

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 the Available_Audio_Contexts characteristics 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. 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 was not possible with Bluetooth Classic Audio profiles.

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 of 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 current resource constraints, such as when an Acceptor is supporting multiple Audio Streams, which may need different parameters. This is where the Audio Stream Control Service comes into play, by defining Endpoints for each Audio Stream. 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. An Acceptor must expose at least one ASE for every Audio Stream it is capable of supporting.

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 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

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.

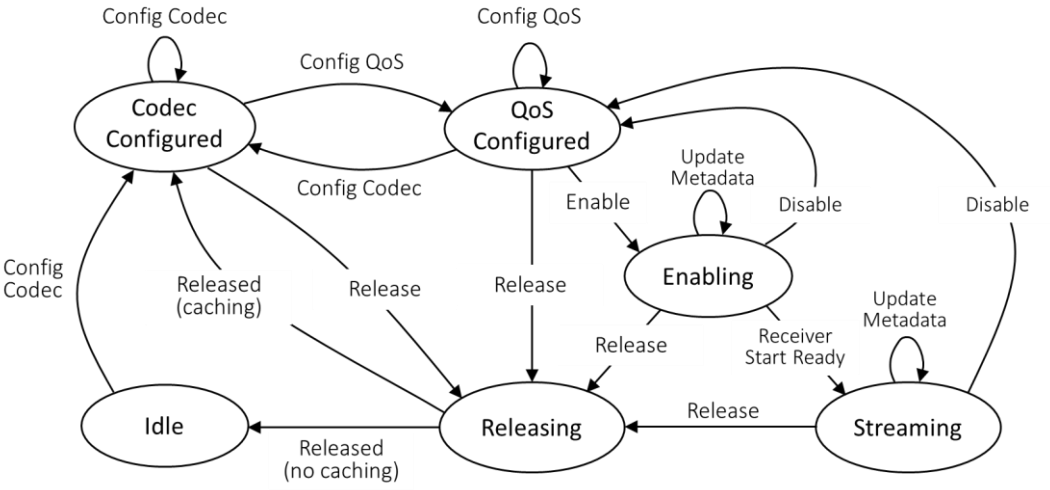


Figure 7.4 The state machine for a Sink ASE

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 ASEs is that an Acceptor supports a separate instance of each ASE 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. 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.

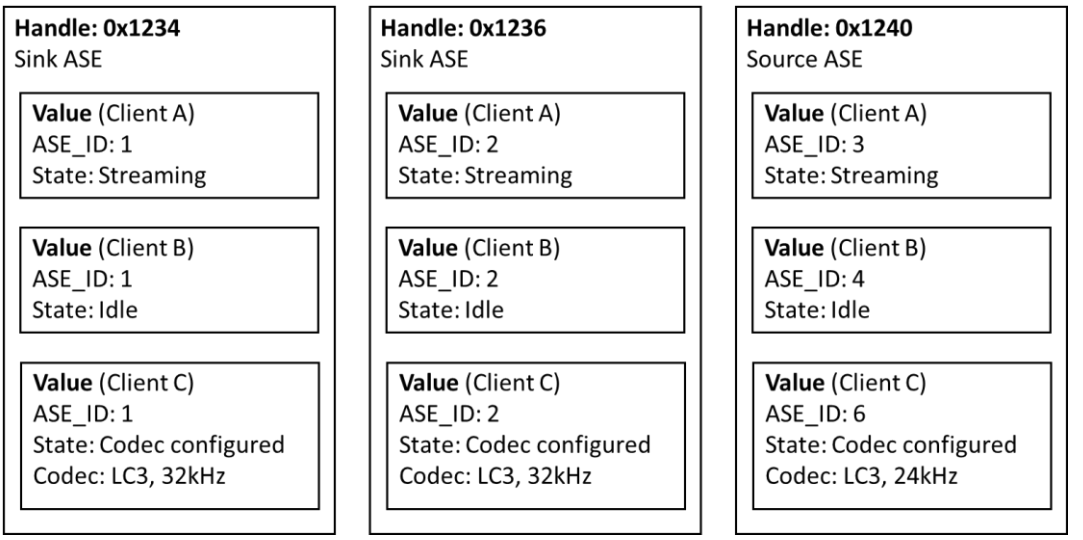


Figure 7.5 An example of exposed ASE states for multiple Clients

Figure 7.5 shows how these principles work. It shows how an Acceptor with three ASEs might expose its instantiated states for three different Clients. Currently, only Client A has established streams with the ASEs which have 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 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 Stream 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. To perform the transitions that move us through the ASE state machine to configure an ASE and enable a CIS, we need to move to the Basic Audio Profile which defines 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 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 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.)

A simplified sequence chart for this process is shown in Figure 7.6. The same process applies to both Sink ASEs and Source ASEs.

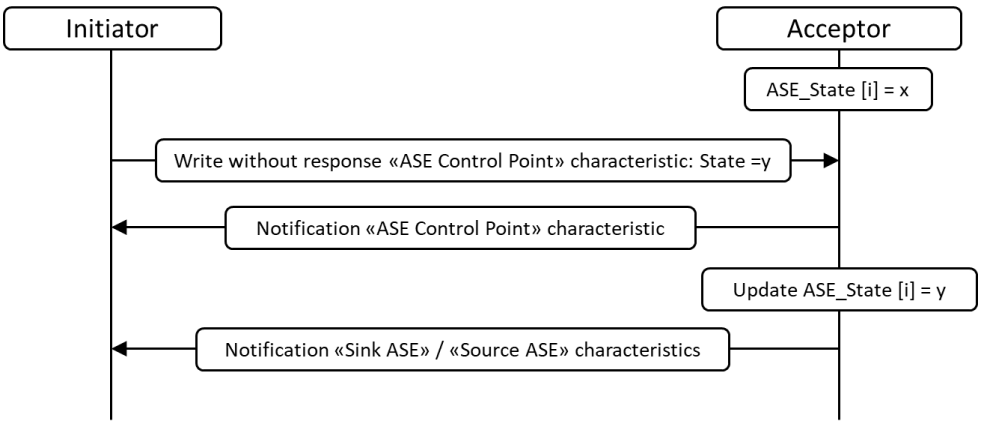


Figure 7.6 Simplified sequence diagram for ASE Control Point operations

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 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.

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.

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 Configuring an ASE and a CIG

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, whether there are mandated features from another top level profile, which means it needs to follow CAP as well, or whether it wishes to use optional or proprietary features. 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

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, so it can jump straight into CAP and run the CAP Connection procedure for non-bonded devices [CAP 8.1.2] to make a connection. 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 7.4.2], 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 confirm that each Acceptor has at least one ASE supporting the correct direction for each Audio Stream it wants to set up. 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].

Audio role discovery and ASE_ID discovery are both mandatory, 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 optional settings.

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. If an Initiator is just using mandatory settings for BAP or a profile which it has already discovered that the Acceptor supports, it may have enough information to skip this stage. Similarly, if the request for connection came from the Acceptor, either directly, or through an Announcement, that may have already provided this information.

The next step is to configure the codec for each ASE. Before doing that, the Initiator should check that each ASE can support the intended use case by checking the 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.

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.

The second 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 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 its HCI command are also 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 the 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.

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 command to instruct the Core to schedule CISes that best meet the application requirements. (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	**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.
Packing	*Preferred packing (sequential vs interleaved)*
Framing	Framed if set to 1, *otherwise up to the Controller on a per-CIS basis.*
Max_Transport_Latency_C_To_P	The maximum transport latency for each direction.
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]	

Parameter	Description
PHY_C_To_P[i]	*Bitfield indicating the possible PHYs to use for each direction. If more than one bit is set, the Controller decides.*
PHY_P_To_C[i]	
RTN_C_To_P[i]	*The Preferred number of retransmissions in each direction. The Controller can ignore this.*
RTN_P_To_C[i]	

Table 7.16 LE Set CIG Parameters HCI parameters

In Table 7.16, the parameters highlighted in italics are recommendations to the Controller, which it may choose to ignore.

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 devices' 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.

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.

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 has actually scheduled.)
CIS_ID [i]	
SDU_Interval [i]	
Framing [i]	
PHY [i]	
Max_SDU [i]	
Retransmission_Number [i]	
Max_Transport_Latency [i]	
Presentation_Delay [i]	Value which the Acceptor will use for rendering or capture.

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 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 latency related to their relative position in the room, but that is not the norm.

If there are one or more Source ASEs, they 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.

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», it should work with the lowest common minimum value.

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	Value which the Acceptor will use for rendering or capture.

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 sent the HCI LE Set CIG Parameters command to its Controller, it moved the CIG to its Configured state. 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 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 Controller is also having to support other wireless connections, it may decide to compromise to allocate airtime between the different and set a lower value than its Host requested. At the point that each CIS is established, the settings will be configured at the Link Layer level and 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. 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.

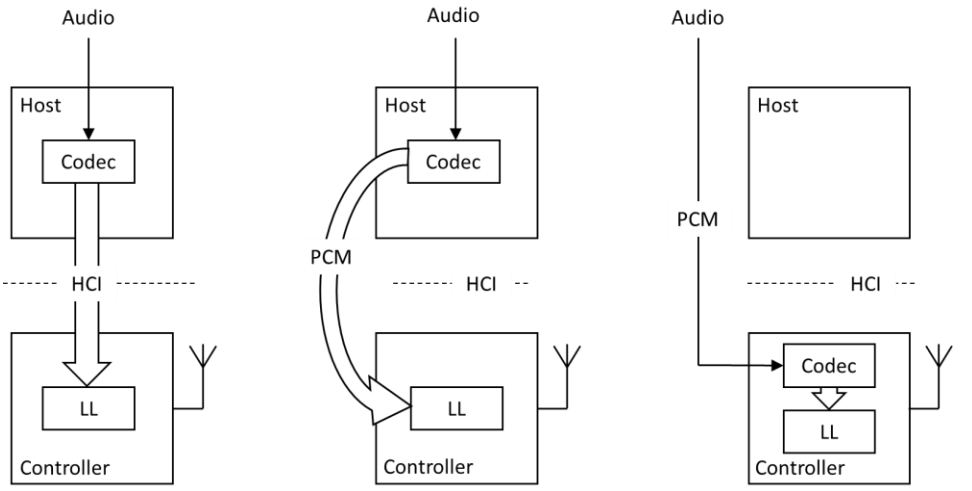


Figure 7.7 Common Audio Data Path configurations

Figure 7.7 shows three of the most common data path configurations, showing that the codec can be implemented in the Host or the Controller. Audio is typically routed from the codec via a PCM interface, but, if encoded in the Host, it 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.

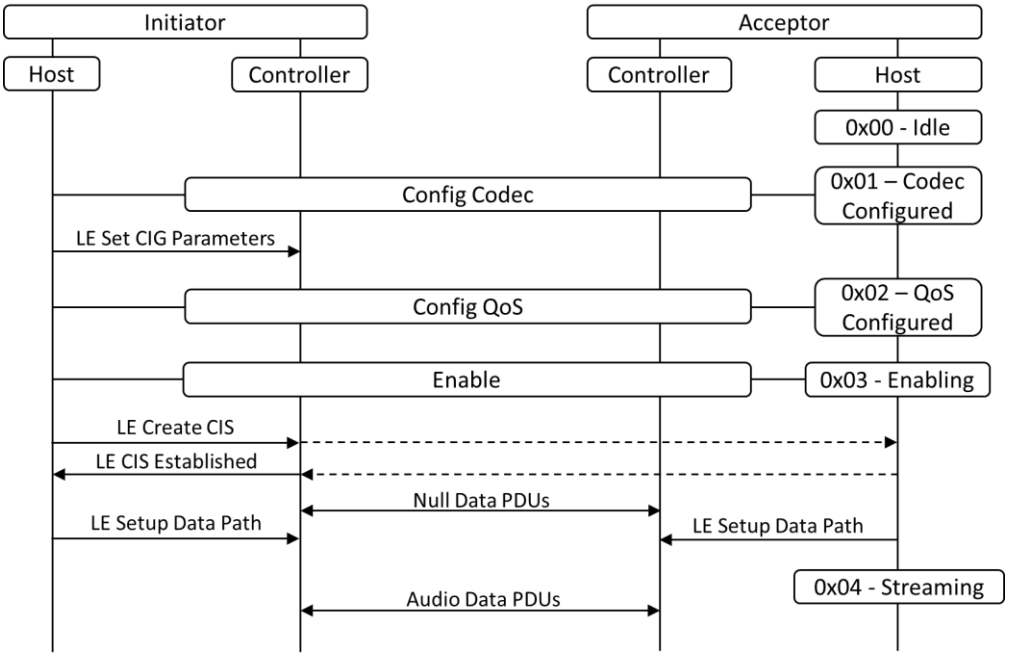


Figure 7.8 Establishment of a Sink ASE, showing the ASE states

Figure 7.8 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 7.9. 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.

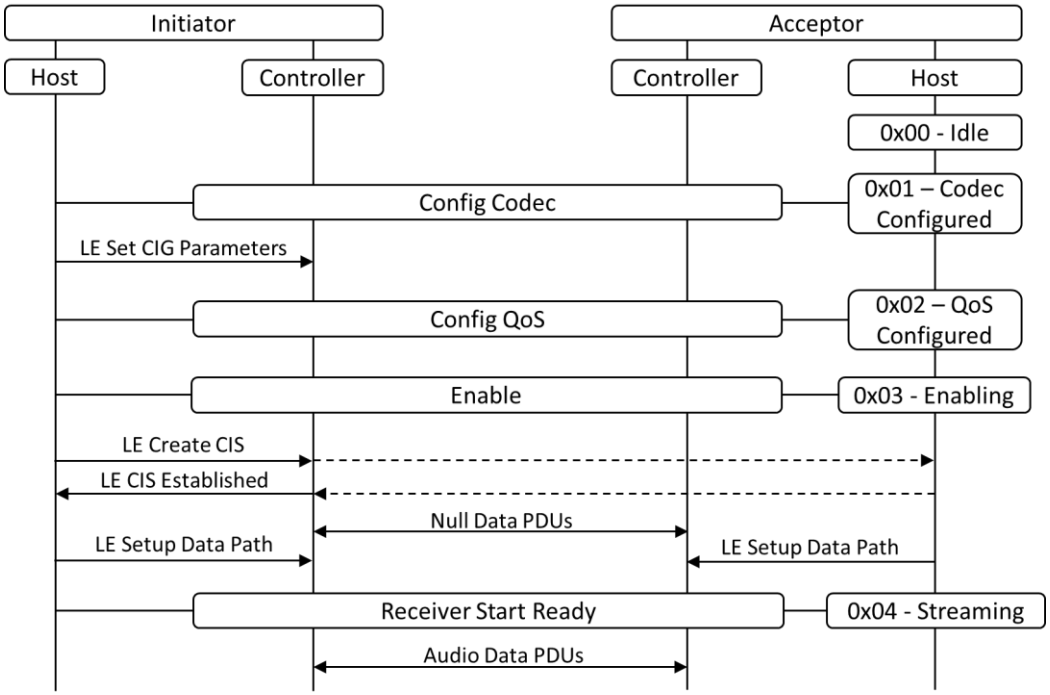


Figure 7.9 Establishment of a Source ASE, showing the ASE states

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.

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 should be moved to the Inactive state.

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 7.10

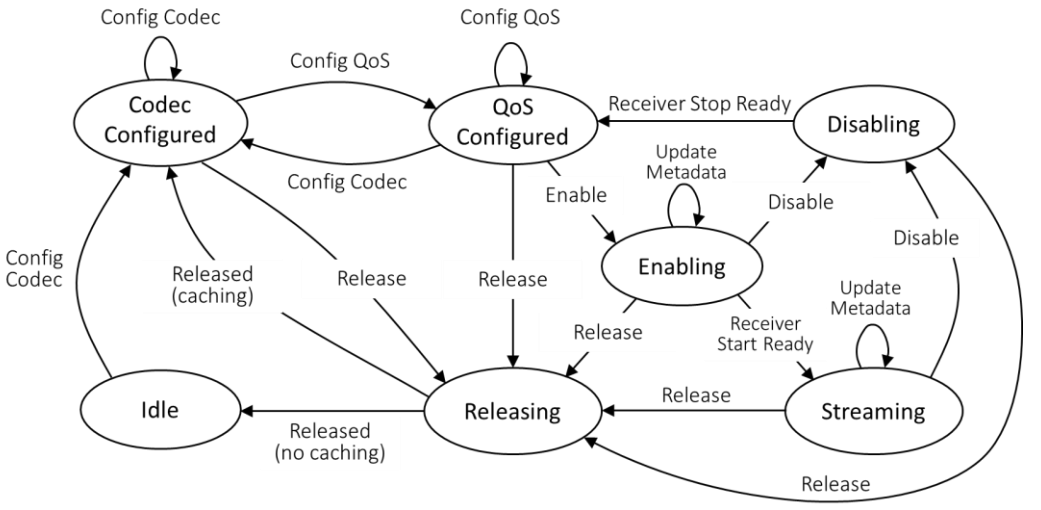


Figure 7.10 State machine for a Source ASE

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 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

7.4.9 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 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 CIS 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, 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, that increases to 90%. By comparison, a bidirectional 32_2_2 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 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. Making this decision will usually depend on other airtime resource demands in the Initiator. 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 hearable 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 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 both of these approaches are unlikely in the short term, as the phone will continue to consider the earbuds as a resource to use as it likes.

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 synchronisation 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.)

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 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.

Once again, the main specification is BAP – the Basic Audio Profile, but 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 synchronise with them.

CAP once again comes into play. As well as providing procedures to set up and receive broadcast streams, it also defines the role of a Commander. A Commander can be a physical device, or 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's main task is to lay down rules about associating Context Types and Content Control IDs and repeat 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 possibilities within broadcast audio, examining some of the new use cases that it can enable.

Although the broadcast topology originated with a desire to improve the quality provided by inductive hearing loops, Bluetooth LE Audio provides far more versatility than an inductive loop. In many cases there will be an ACL connection between the Broadcast Source and the Broadcast Receiver in addition to the one between the Broadcast Source and a Commander. Devices can even support both broadcast and unicast at the same time, acting as a relay between broadcast and unicast connections. But before we get into those complications, we'll start with the basics of broadcast by itself.

The broadcast specifications fall into three separate functions:

- Transmitting a broadcast Audio Stream. At its simplest, a Broadcaster (which is an easier name to use for a Broadcast Source) acts independently – 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, (although technically that's just a role, of which the Broadcast Assistant is a sub-role). 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 synchronise to it. At this basic level, it also acts independently. A Broadcast Receiver can synchronise 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 devices. Instead, the Broadcast Source is driven entirely by its specific application.

One result of this is that a Broadcast Source has a very simple state machine, shown in Figure 8.1. 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.

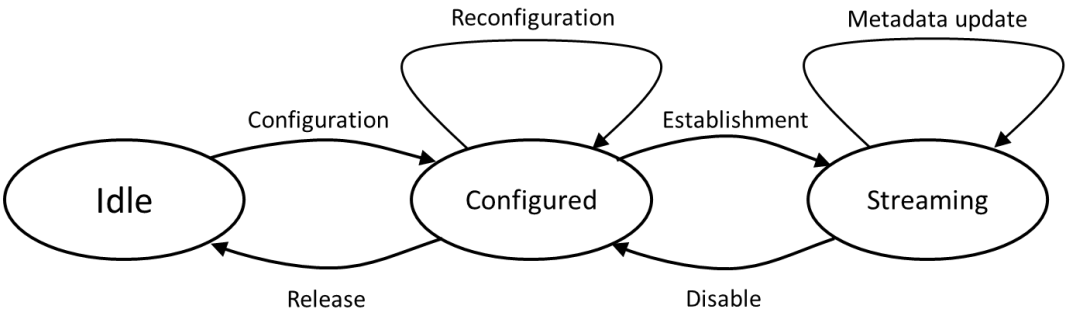


Figure 8.1 Broadcast Source State Machine and Configuration procedures

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. It also needs to provide the 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.

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 when the Initiator is ready to start broadcasting. CAP bundles them together into just five procedures, combining configuration and establishment into its Broadcast Audio Start procedure:

- Broadcast Audio Start procedure
- Broadcast Audio Update procedure
- Broadcast Audio Stop procedure
- Broadcast Audio Reception Start procedure
- Broadcast Audio Reception Stop procedure

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 could be used 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.

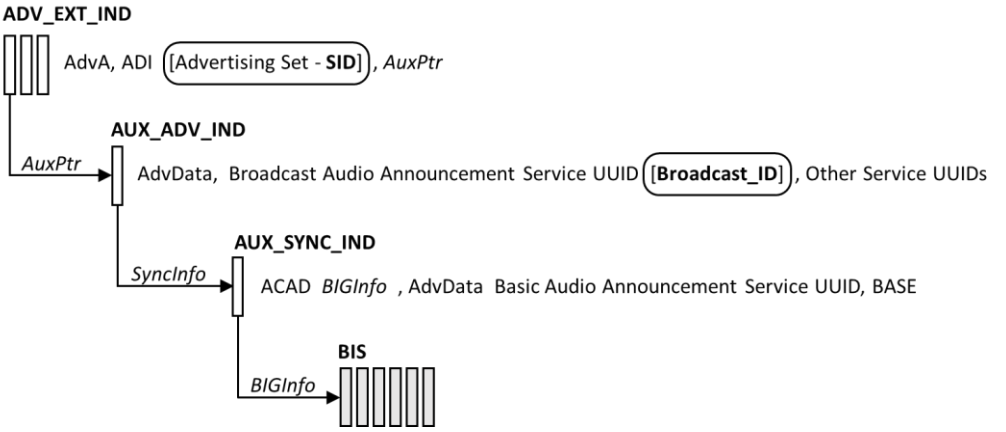


Figure 8.2 Data required to set up a Broadcast Source

Figure 8.2 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 Audio Stream is to assemble all of the data needed for the Controller to populate the Broadcast Audio Announcement Service, BIGInfo and BASE, which is 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	Inherited from PBP
TMAP	None	
PBP	None	ProgramInfo is recommended to describe 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 QoS settings.

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 8.3.

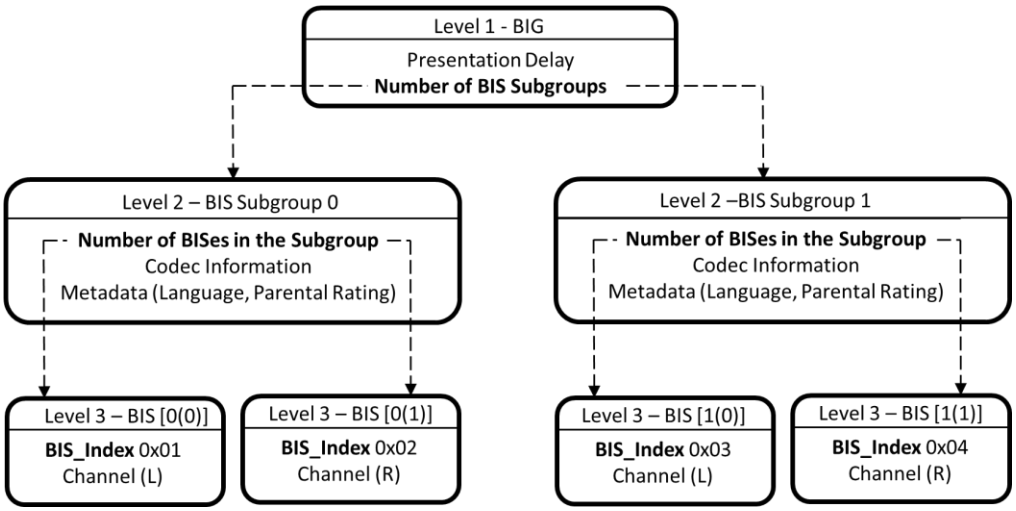


Figure 8.3 Simplified BASE structure

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 µs (0x000000 to 0xFFFF)
Num_Subgroups	1	The number of subgroups used to group BISes with common features in this BIG.

Parameter	Size (Octets)	Description
Level 2 – BIS Subgroup Parameters (common parameters for subgroups of BISes)		
Num_BIS[i]	1	The number of BISes in this subgroup.
Codec_ID[i]	5	Codec information for this subgroup. Typically, this will be LC3 (0x06).
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.
Level 3 – Specific BIS Parameters (if required, for individual BISes)		
BIS_index[i[k]][44]	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.

Table 8.2 The parameters of the BASE.

[44] One point of possible confusion to be aware of is that BIS_index starts from 1, but CIS_IDs run from 0.

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 BISes 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 or joint stereo stream. 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 kHz. But that is the only thing which changes at Level 3. Different languages, different parental ratings, etc., all need to have their own subgroup defined at Level 2.

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.

Level	Parameter	Value
2	Num_BIS [0] Codec_ID [0] Codec_Specific_Configuration LV [0] Metadata LV [0]	Num_BIS = 2
3	BIS_index [0[0]] Codec_Specific_Configuration LV [0[0]]	0x01
2	Num_BIS [0] Codec_ID [0] Codec_Specific_Configuration LV [0] Metadata LV [0]	Num_BIS = 2
3	BIS_index [0[1]] Codec_Specific_Configuration LV [0[1]]	0x02
2	Num_BIS [1] Codec_ID [1] Codec_Specific_Configuration LV [1] Metadata LV [1]	Num_BIS = 2
3	BIS_index [1[0]] Codec_Specific_Configuration LV [1[0]]	0x03
2	Num_BIS [1] Codec_ID [1] Codec_Specific_Configuration LV [1] Metadata LV [1]	Num_BIS = 2
3	BIS_index [1[1]] Codec_Specific_Configuration LV [1[1]]	0x04

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 Parameters, 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	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 1, 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

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 every BIS is the same size, which must fit the largest SDU. Different BISes within the BIG may 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.

Having created the BASE structure, the Host can ask the Controller to schedule the BIG and BISes, using the LE_Create_BIG command (defined in the 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 reflect what is in 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
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 msecs.	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 Audio Source enters the Configured State. At this stage it is not yet transmitting any audio data.

8.2.3 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 BAP Modifying Broadcast Sources procedure [BAP 6.5.5]. The Broadcaster does not need to stop the Audio Stream, but can make the change dynamically when in the Configured or Streaming States.

There is no guarantee that an Acceptor which is receiving the Audio Stream will see such a change. Once an Acceptor has synchronised 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.

8.2.4 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 Establishment[45] procedure [BAP 6.3.2].

First, it 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.

[45] This should not be confused with the BASE. It's unfortunate it has the same initials.

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. If it has no connections, it will never know whether any device detects or synchronises to those broadcasts.

8.2.5 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 and the BIGInfo. Synchronised 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 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.

8.2.6 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

If a Broadcast Sink wants to receive a broadcast Audio Stream, it needs to scan to find it. The same procedures for finding it 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 Commander can do 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 an Acceptor scanning on its own is essentially operating at the BAP level.

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 ones of interest are Extended Advertisements which contain the Broadcast Audio Announcement Service UUID with a Broadcast_ID. The Extended

Advertisements may also include other Service UUIDs, such as those defined in the Public Broadcast Profile, which the scanner can use to filter the streams which it wants to investigate.

Having found an appropriate Extended Advertisement, the scanner will look for the SyncInfo data and use this to synchronise to the Periodic Advertisements associated with the Broadcast_ID. Once synchronised, the scanner can find and parse the data in the BASE, which provides information about the set of 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 about the available Broadcast Sources within range which have Audio Streams of interest and decides for itself which one to receive. This is entirely driven by the implementation. It may look for a known Broadcast_ID, which it has connected to before, or information in the BASE metadata. It could connect to the first stream it finds, and then step through any other streams which are available, or it could collect data from every Broadcast Source within range and present this to the user in some form to allow them to make a manual choice. All of that is implementation specific.

The Acceptor can use additional information to inform its choice. It would not normally bother with streams if their Context Type did not match any of the Acceptor's Available Context Types. Nor would it want to receive a stream for an Audio Location that it does not support. But the decision is the Acceptor's. Without a Commander, or a proprietary link between two Acceptors, 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 Commander (or a Commander application), or implement a proprietary link between left and right earbuds. However, 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 / Commander comes in.

8.3.2 Synchronising to a broadcast Audio Stream

Once it has decided on a Broadcast Source to receive, an Acceptor 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 Acceptor's Audio Location. The BIS_Index for each BIS defines the order of the BISes within the BIG. Knowing this, the Acceptor can determine which BISes in the BIG it wants to receive.

The Acceptor 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 synchronisation to a broadcast Audio Stream

If an Acceptor decides it wants to stop receiving a broadcast Audio Stream it acts autonomously to disconnect the Audio data path and stops synchronisation with the PA and the BIS.

8.4 The broadcast reception user experience

Although an Acceptor is allowed to acquire a Broadcast Stream by itself, it's not 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 many 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 a lot 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 together, and always connecting both earbuds to the same Broadcast Source.

So, although the explanations above are valid and indicate how to receive a broadcast stream, they're largely academic. 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 the broadcast streams and telling one or more Acceptors 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 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 to assist the distribution 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 States for notifications, which means that all of them will be updated whenever a change is made to the Broadcast State, either by a Broadcast Assistant writing to it, or as a result of a local action on the Acceptor.

The Broadcast Receive State characteristic exposes what the Acceptor 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 synchronise to. In many cases, that will be just one. Now on to the Commander.

8.6 Commanders

Although most people see Broadcast as the big new feature of Bluetooth LE Audio, the most important innovation is probably the concept of the Commander, which provides remote control and management of broadcast Audio Streams. In a new world of multiple broadcast streams, Commanders 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.

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 are encrypted, requiring methods to acquire the encryption keys. Once again, that is enabled by Commanders.

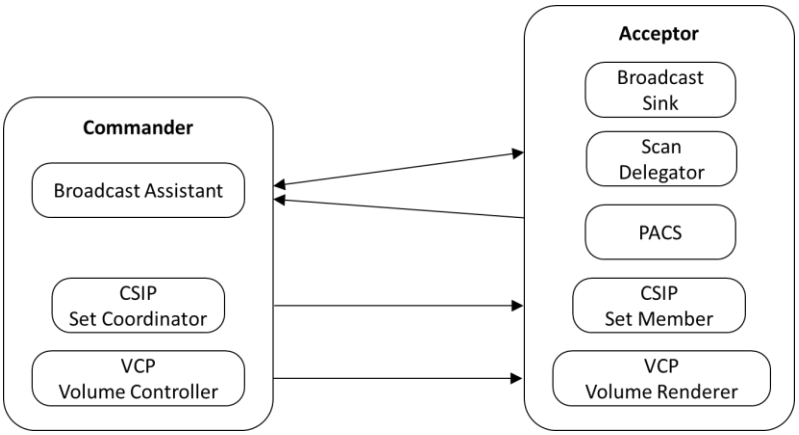


Figure 8.4 The major elements of a Commander and broadcast Acceptor

Figure 8.4 illustrates the major components of a Commander and the main broadcast-related parts of an Acceptor. Both are roles defined in CAP, but I'm using them more generally in this book. It's useful to visualise a Commander as a device in its own right, like a TV remote control, because for a user, what it does is very similar. Commanders 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's information about a Broadcast Source, or volume control.
- 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 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. They 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.

8.6.1 Broadcast Assistants

Broadcast Assistants allow other devices to perform 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, scanning on behalf of a Broadcast Sink. 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 zinc-air 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 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 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. 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.

8.6.2 Scan Delegators

Commanders 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 other devices taking the Broadcast Assistant Role, which can take over the job of scanning for Broadcasters. 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.

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.

Once the Broadcast Assistant has detected a suitable Broadcast Source, it can inform the Scan Delegator by writing to one of its Broadcast Receive State characteristics 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 on a Commander, 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 modify the appropriate Broadcast Source value using the Modify Broadcast Source procedure [BAP 6.5.5]. It should also check that the Broadcast Sink is capable of decoding any proposed Audio Source.

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
Source Information			
	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
Action			
	PA_Sync	1	0x00: Don't synchronise 0x01: Synchronise using PAST 0x02: Synchronise 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 synchronise to BIS_Index[i[k]] 0x01: Synchronise to BIS_Index[i[k]]
Configuration			
	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 synchronise 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 Commander to the Acceptor 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, as the whole point of a subgroup is to arrange associated streams together. If two subgroups carried different languages, say Japanese and German, the expectation is that you would only choose one.

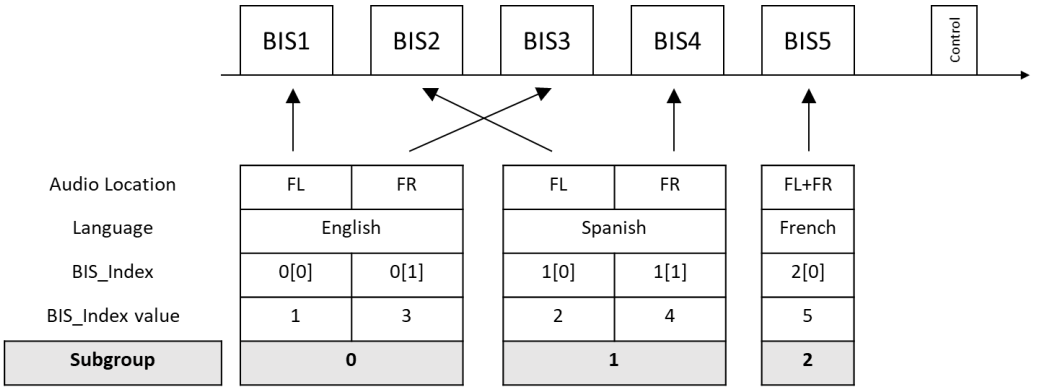


Figure 8.5 Illustration of mapping from BIS number to BIS_Index

Figure 8.5 illustrates how BIS_Index values in subgroups relate to the ordering in a BIG, where BISes are arranged in numeric order. In this case, those don't align with the BIS_Index values, hence the mapping of BIS_Index. In writing a BIS_Sync value, only one subgroup can be selected. So, if subgroup 0 were selected from Figure 8.5, the only allowable values for BIS_Sync[0] would be that one or both of bits 0 and 1 are 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.

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 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 connection information which is conveyed in these two operations, it is entirely up to the Acceptor whether it follows the request to synchronise with the PA and the BISes which are written to it. That's up to the implementation.

If the Acceptor decides to act on the instruction from the Commander, it will use the instructions to synchronise to the Periodic Advertising train, either by using PAST, or scanning with the Broadcast Source information it's been given, retrieving the BASE and BIGInfo from the ACAD and Basic Audio Announcement in the PA. These give it all of the information it needs to synchronise to the BIG, after which it will use the BIS_Sync value to select which specific BIS or BISes it receives.

PAST – the Periodic Advertising Sync Transfer procedure [Core, Vol 6, Part B, Section 9.5.4] is the most efficient method, as the Broadcast Assistant provides the Scan Delegator with the SyncInfo data allowing it to synchronise directly to the PA. This process is known as Scan Offloading.

Disconnection can be performed autonomously by the Acceptor, or a Commander can use the Modify Source operation to set the appropriate BIS_Sync bit to 0b0. It may also tell the Acceptor to stop synchronising 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 precis of the different IDs involved in these operations, as they can be confusing.

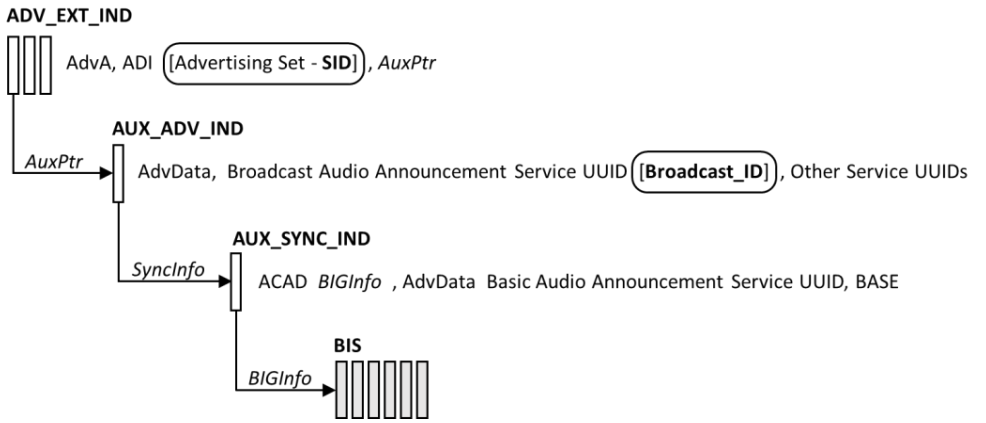


Figure 8.6 Set and Broadcast IDs

Referring to Figure 8.6, the Set and Broadcast IDs are properties of the Broadcast Source. Both are randomly generated numbers. 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. 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.

The Broadcast_ID is specific to a BIG and is carried in the Broadcast Audio Announcement Service UUID. It remains static for the lifetime of the BIG.

The Source_ID is an Acceptor generated number which 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.

8.6.4 The Broadcast Receive State characteristic

On the Acceptor, the Broadcast Receive State characteristic is used to inform any Broadcast Assistant of its current status regarding broadcast Audio Streams. Its structure is shown in Table 8.8.

Field	**Size (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
PA_Sync_State	1	Current synchronisation status to the PA: 0x00 – Not synchronised to the PA 0x01 – SyncInfo request 0x02 – Synchronised to the PA 0x03 – Synchronisation failed 0x04 – No PAST Other values – RFU

Field	Size (Octets)	Description
BIG_Encryption	1	Encryption status: 0x00 – Not encrypted 0x01 – Broadcast_Code required 0x02 – Decrypting 0x03 – Bad_Code (wrong encryption key)
Bad_Code	Varies	If BIG_Encryption = 0x03 (wrong key), this contains the value of the current, incorrect key. Otherwise not present.
Num_Subgroups	1	Number of subgroups
BIS_Sync_State[i]	4	BIS synchronisation 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 an operation and status, and also be notified by the Server to denote than an operation is complete, such as synchronising to a PA, or to request an action from a Client. When they're being notified, some of these act as error reports, while others signal a request to a Broadcast Assistant. Amongst these, the most important are notifying a PA_Sync State of 0x01 to request SyncInfo and notifying a BIG_Encryption state of 0x02 to request a Broadcast_Code. Any autonomous change of BIS synchronisation or loss of BIS can be notified through the updated BIS_Sync_State.

8.7 Broadcast_Codes

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 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, tablets and laptops, can collocate a Broadcast Assistant with the Broadcast Source. The Assistant then has direct access to the Broadcast_Codes and can write them directly to the Broadcast Sink. It requires the Acceptors 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 means that Broadcast Sources and Commanders need to support a variety of different methods to acquire Broadcast_Codes.

8.8 Receiving Broadcast Audio Streams (with a Commander)

Having learnt about Commanders, we can revisit how devices are likely to receive Broadcast Streams in the real world, which will almost always involve a Commander, 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

The first job is for the Scan Delegator to find itself some Broadcast Assistants, which it does by sending out Extended Advertisements which contain a Service AD Data Type containing a Broadcast Audio Scan Service UUID. These are called Solicitation Requests [BAP 6.5.2].

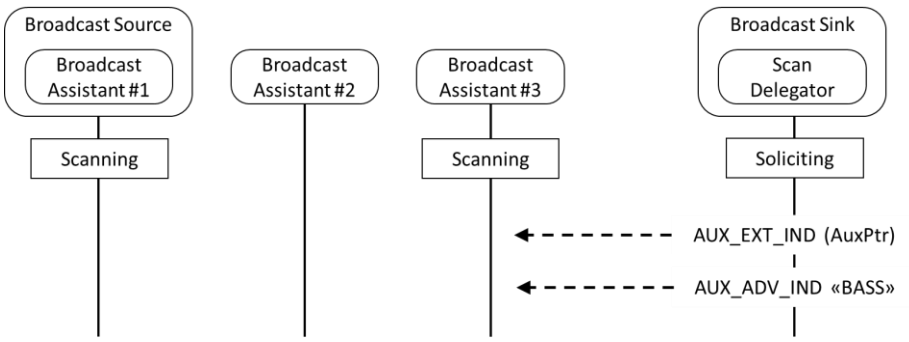


Figure 8.7 The Scan Delegator solicitation process

Figure 8.7 shows a Scan Delegator sending out Extended Advertisements containing the BASS Service UUID. To the left, three Broadcast Assistants are within range. Broadcast Assistant #1 is collocated with a Broadcast Sink 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 Assistant #1 and #3 should respond to the Solicitation requests.

CAP, as always, tells each Broadcast Assistant to start off by connecting, establishing the members of a Coordinated Set, 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 accept. 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, requires encryption, etc., etc.

Only one Acceptor of a Coordinated Set needs to perform the Solicitation process. 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.

Figure 8.7 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 States 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 synchronised 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.

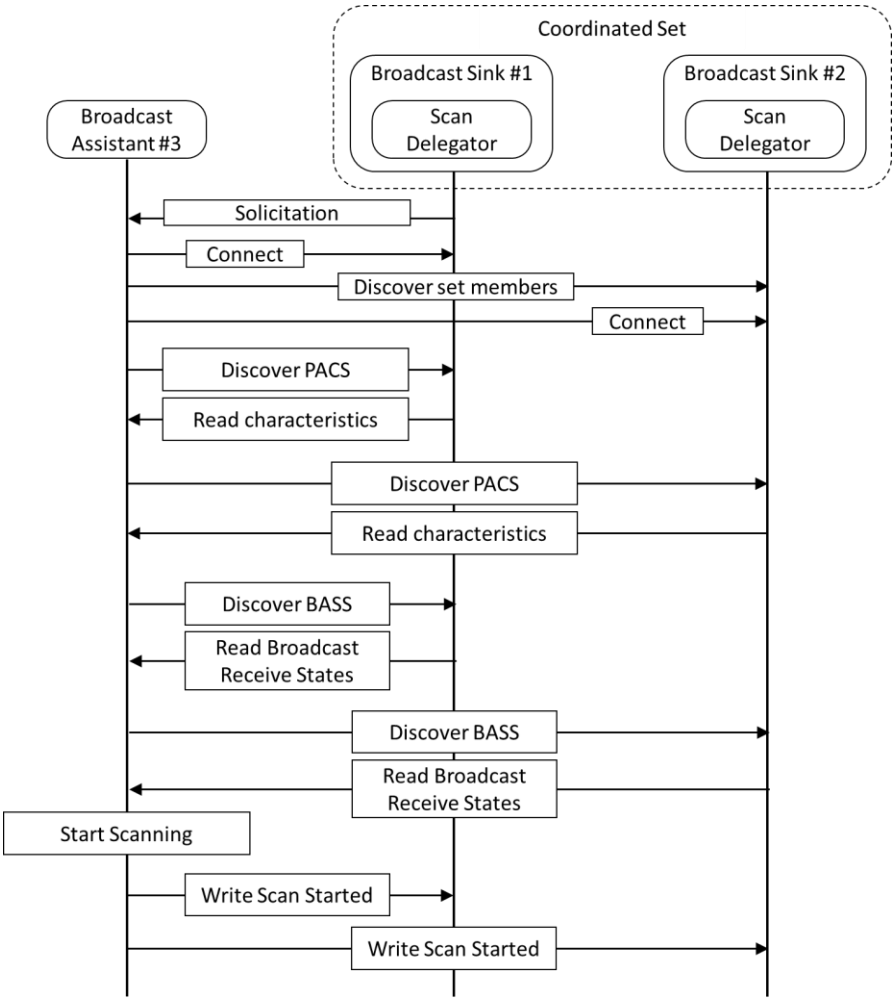


Figure 8.8 Discovery of Broadcast Sink Capabilities and State by a Broadcast Assistant

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 probably be presented in the form of a sorted list, from which the user can select a broadcast Audio Stream to connect to.

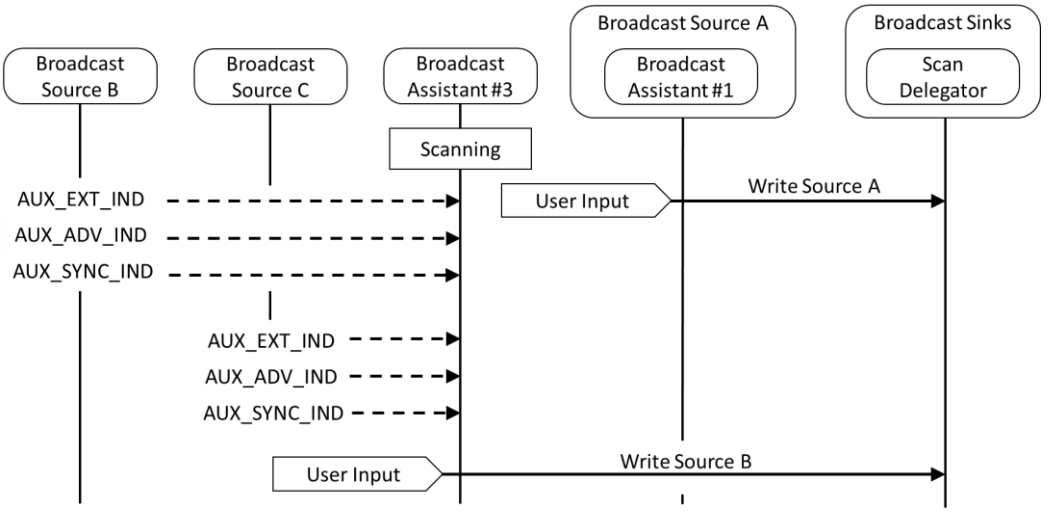


Figure 8.9 Remote broadcast scanning

Figure 8.9 follows on from Figure 8.8, where Broadcast Assistant #3 is scanning on behalf of the two Broadcast Sinks (shown here as a single entity). It has discovered Broadcast Source B and Broadcast Source C. The user makes a decision on the Commander to listen to Broadcast Source C and selects it, at which point Broadcast Assistant #3 writes the information about Broadcast Source C to the BASS instances in the Scan Delegators of the two Broadcast Sinks.

Figure 8.9 also shows how a collocated Broadcast Assistant can behave. In this case, Broadcast Assistant #1 is collocated with Broadcast Source A. Broadcast Assistant #1 has never written to the Broadcast Scan Control Point to say it is scanning on behalf of the Scan Delegator, as its only purpose is to provide the Scan Delegator with information about its collocated Broadcast Source. As soon as it connects to the Scan Delegator it can write that information to a Broadcast Receive State. (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.)

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 to the user.

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 a Broadcast Source, which we described in Section 8.6.3. In most cases, when the user selects a Broadcast Source on their Commander, the Broadcast Assistant will not just write the details of that source to the Broadcast Audio Scan Control Point characteristic, but will also set the PA_Sync and BIS_Sync parameters to instruct each Acceptor to start receiving the relevant streams.

As the two Acceptors are acting independently, they may take different amounts of time to obtain the PA train before they can synchronise 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 the two earbuds. The Bluetooth specifications don't contain any details on how to address this, but one method is for the Commander to do this in two stages, starting by setting the PA_Sync bits, waiting for notification that both Acceptors are synchronised and only at that point sending a Modify Source operation, instructing them to synchronise their BISes and receive and render audio.

8.8.4 Broadcast_Codes (revisited)

In Chapter 12 we will see just how important Broadcast_Codes are for new audio applications. If we look back at Figure 8.9, and assume that the broadcast Audio Streams from both Broadcast Source A and Broadcast Source C are encrypted, there's likely to be a slightly different way in which the two Broadcast_Codes would be obtained.

Broadcast Source A's Broadcast Assistant knows the Broadcast_Code for the streams – that's why it's collocated. So as soon as the user selects Broadcast Source A, the Broadcast_Code will be sent over in the Add Source Command.

In the case of Broadcast Source C, the Broadcast Assistant may not know the code, particularly if it has not been paired with Broadcast Source C. In this case, one of the Scan Delegators can ask all of its Broadcast Assistants if they know the Broadcast_Code, by notifying the appropriate Receive State characteristic, with the 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, and include that value in the Bad_Code field, so that a Broadcast Assistant with the same bad code doesn't waste time resending it. 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.

As we'll see in Chapter 12, there is a range of out-of-band methods by which a Broadcast Assistant may obtain a Broadcast_Code, opening up some interesting new use cases. Some of these may directly trigger the broadcast Audio Stream acquisition process.

8.8.5 Ending reception of a broadcast Audio Stream

Reception 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 synchronised 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 should perform the Remove Source Operation (0x05) [BASS 3.1.1.7] to remove the associated Broadcast Receive State. Note that there is no state machine for the Broadcast Sink. It uses its Broadcast Receive State characteristic to synchronise 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 not the same as the general case of changing from a unicast use case to broadcast use case, but is specific to an application like 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 the Initiator, with the intention that the handover could be seamless with no noticeable break in the stream. (In most cases that will not be the case, 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, although that may be accomplished by an audible tone or message. (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, they would still maintain the 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.

8.10 Presentation Delay – setting values for broadcast

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, so this is probably a default value which many Initiators will use. However, it starts to introduce 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 20ms. As most Acceptors are expected to be qualified to TMAP or HAP (most will probably support both), setting 20ms for a Broadcast Source seems a reasonable compromise. However, a BAP only compliant device 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 guideline offer a pragmatic approach.

- If the Presentation Delay is lower than an Acceptor can accommodate, it should use a value of 40ms (which all devices must support).

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. 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 will 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.

Chapter 9. Telephony and Media Control

After the complexity of setting up unicast and broadcast streams, the four specifications covering telephony and media control 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 pandemic and the associated growth in home working have 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 borrow it on demand. That means that the traditional controls of Stop, Play, Fast Forward and Fast Reverse need to be supplemented with new controls which move beyond the physical device to the source of the audio.

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, 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 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 MCP.

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 a 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 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 as it wishes.

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 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 to map incoming commands from a Client to the appropriate application.

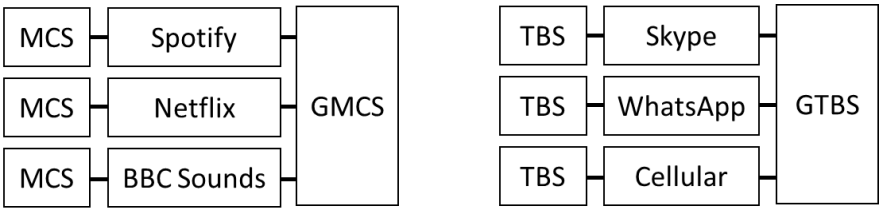


Figure 9.1 Representation of Control Services and their generic counterparts

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, with 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.

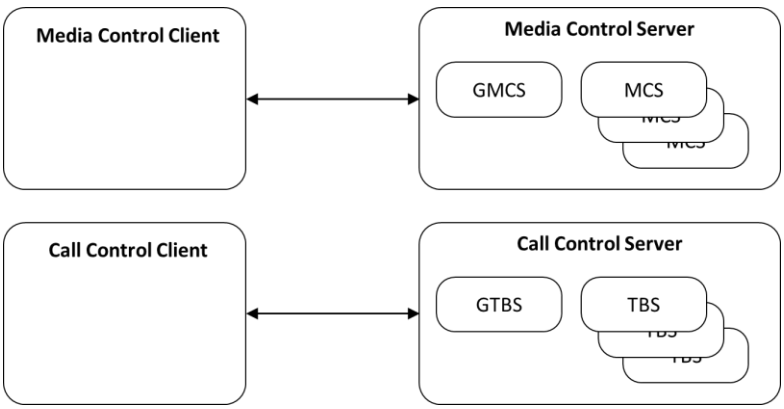


Figure 9.2 Relationship of generic and specific control services

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 by far 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 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 is held on the device where the audio originates – the media player for MCS, or the telephony device, acting as a gateway to the call bearer for TBS, but 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, the Server notifies its new state. It's exactly the same principle we came across in the ASCS state machine - there is one device which has the state, and 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.

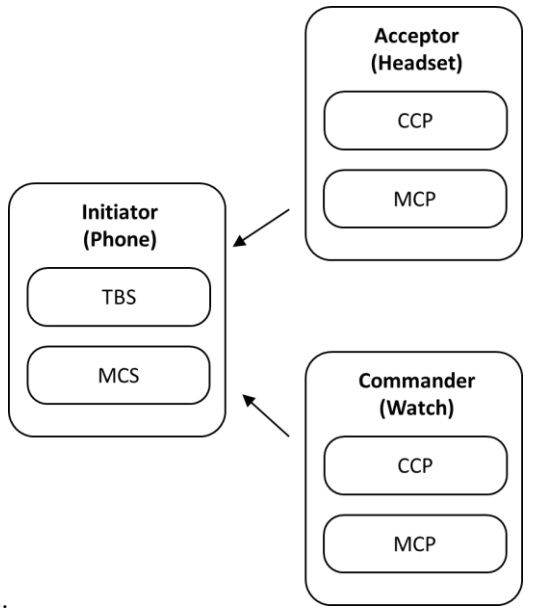


Figure 9.3 Topologies for content control

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.

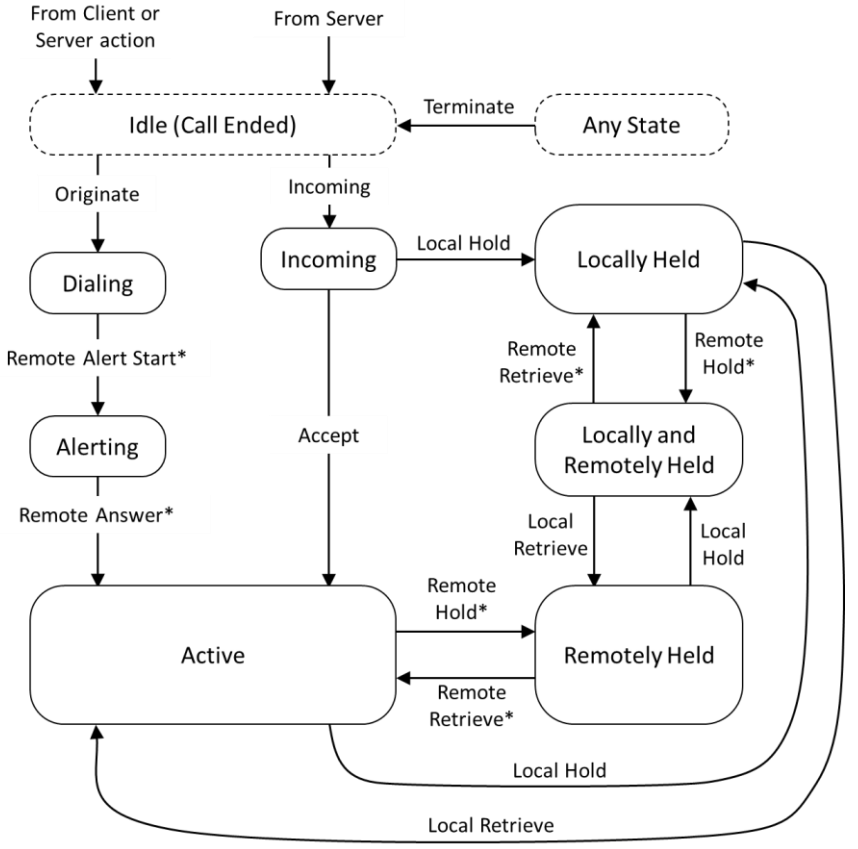


Figure 9.4 The TBS state machine (asterisks denote remote operations)

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. 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 external 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 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.

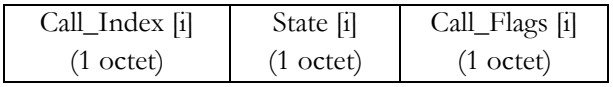


Figure 9.5 Call State characteristic

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

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.

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.

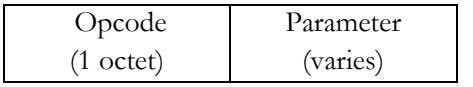


Figure 9.6 The TBS Call Control Point characteristic

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 telephony 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, some phone services do not allow a call to be dialled if a current call if active. Equally, if there is no cellular signal, an attempt to originate a call will fail.

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. 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

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, which is a poor user experience.

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 and the phone (which is a different Initiator) will establish a bidirectional Audio Stream to carry the call.

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.

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 determine 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.

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.

9.3.4 Terminating 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.

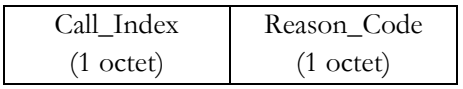


Figure 9.8 Termination Reason characteristic

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

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 (Optional)	Signal strength from 0 (no signal) to 100
Bearer Signal Strength Reporting Interval	How often the signal strength is reported
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.

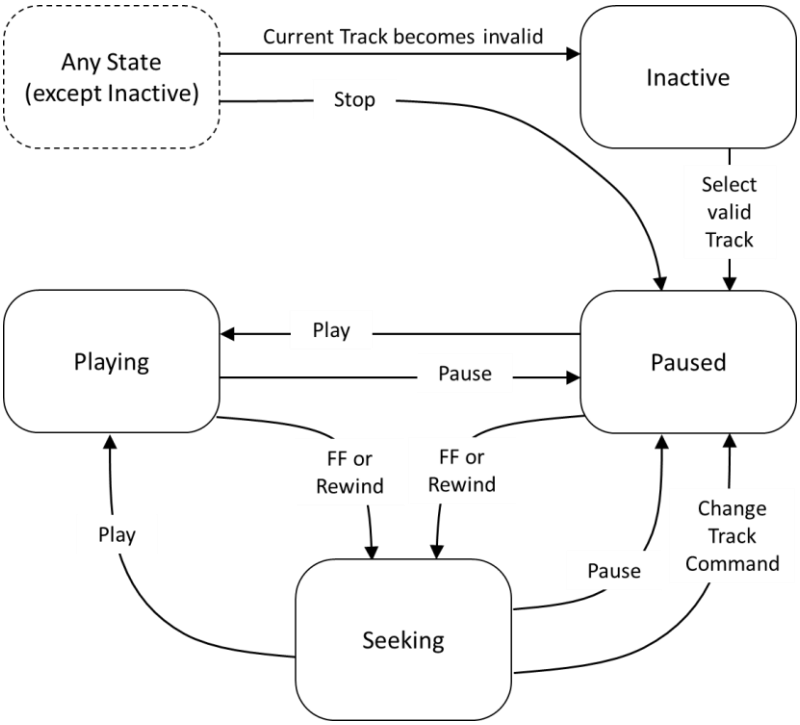


Figure 9.9 MCS state machine for media control

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 Paused by stopping it, or be moved to the Seeking state by issuing a Fast Forward or Fast Rewind command. There is no Stop state or command in the state machine. For digital media, there is no real distinction between the Stop and Pause command. 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 the same.

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.

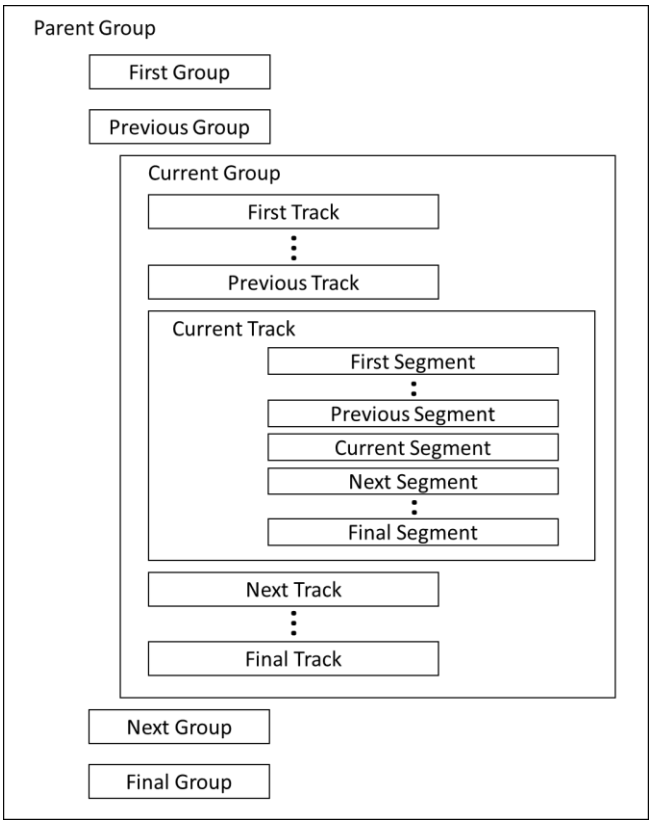


Figure 9.10 MCS' arrangement of Groups and Tracks

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.

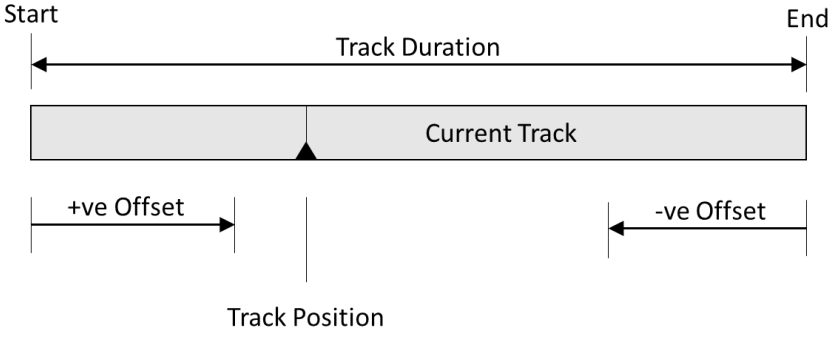


Figure 9.11 Key elements of a Track

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 MSC 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's empty, but is 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 months[46].)

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

Once the Current Track starts playing, the Server should notify the Track Position characteristic, to inform Clients of the current track position. The frequency of notification

[46] This might seem ridiculous, but the longest single video, "Level of Concern" by the group "twenty 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.

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.

Issuing the Stop command will result in the current track becoming invalid, so before any of the commands in Table 9.7 can be used again, the user will need to select another track as current.

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 item opcode n-1 times. A negative value is equivalent to going to the last item and executing the previous item opcode n-1 times.

Moving to a new track makes it current and places it in the Paused state of the state machine of Figure 9.9.

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 Reverse. 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. 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
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.

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.

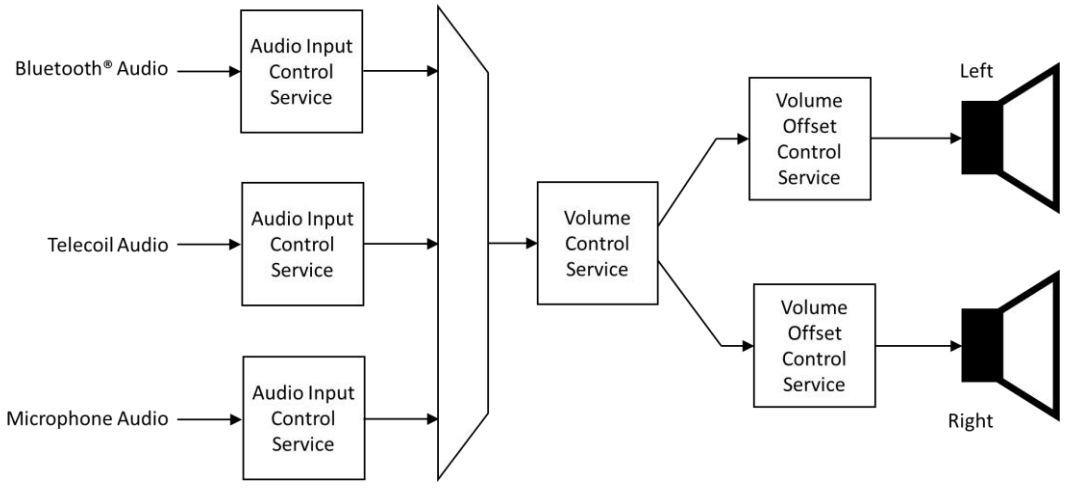


Figure 10.1 Typical implementation of Volume and Audio Input services for a headphone

Figure 10.1 shows a typical implementation for a pair of headphones, or banded hearing aids, which has three possible audio inputs – a Bluetooth Audio Stream, a Telecoil audio input and a microphone input. These inputs are mixed together, using the Audio Input Control Service to set their individual gains and to selectively mute and unmute them. The resulting stream 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 each speaker to accommodate different levels of hearing loss in the left and right ear.

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 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.

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 Figure 10.2. 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

Figure 10.2 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.

The value of the Volume_Setting field is changed as a result of an operation on the Volume Control Point characteristic (q.v.) 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.

Change_Counter is the feature mentioned above, which ensures that all Volume Control Clients are synchronised. 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 what they are currently. 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. The command is ignored, and no notification will be sent. In this case, the Client should recognise the lack of a notification, read the Volume State characteristic and try again.

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 10.1.

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

Opcode	Operation	Operands
0x06	Mute	Change_Counter
0x07-0xFF	Reserved for Future Use	

Table 10.1 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-one[47] 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 ÷ 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: Volume_Setting = Max (Volume_Setting – Step Size, 0)

Relative Volume Up: Volume_Setting = Min (Volume_Setting + 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 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. This can improve the user experience, by starting with the same volume settings which had previously been used.

[47] 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.

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 need to include an instance of the Volume Offset Control Service for every rendered Audio Location. All of the characteristics are accessed using procedures in the Volume Control Profile.

The Volume Offset Control Service includes four characteristics, shown in Table 10.2. 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 10.2 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 10.3 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 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, maintain their 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 10.4 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.

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. 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 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.

An instance of the Audio Input Control Service is normally included for each separate audio path which is intended to be rendered on that device. If there is a single Bluetooth LE Audio path, it offers no advantages over VCS, so does not need to be included. 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 10.5. 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 10.5 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 10.6.

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 10.6 Fields of the Gain Setting Properties characteristic (read only)

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 10.7.

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 10.7 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 10.8.

Name	Size (octets)
Gain_Setting	1
Mute	1
Gain_Mode	1
Change_Counter	1

Table 10.8 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 10.9

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 10.9 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 10.10.

Parameter	Size (Octets)	Value
Opcode	1	0x01
Change_Counter	1	0 to 255
Gain_Setting	1	-128 to 127

Table 10.10 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 10.11.

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 10.11 Format for opcodes 0x02 - 0x05 for Audio Input Control Point characteristic

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 UI. Typical values (in English) are Microphone, HDMI, Bluetooth, etc. 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 10.3 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), whilst VCS provides a global mute function.

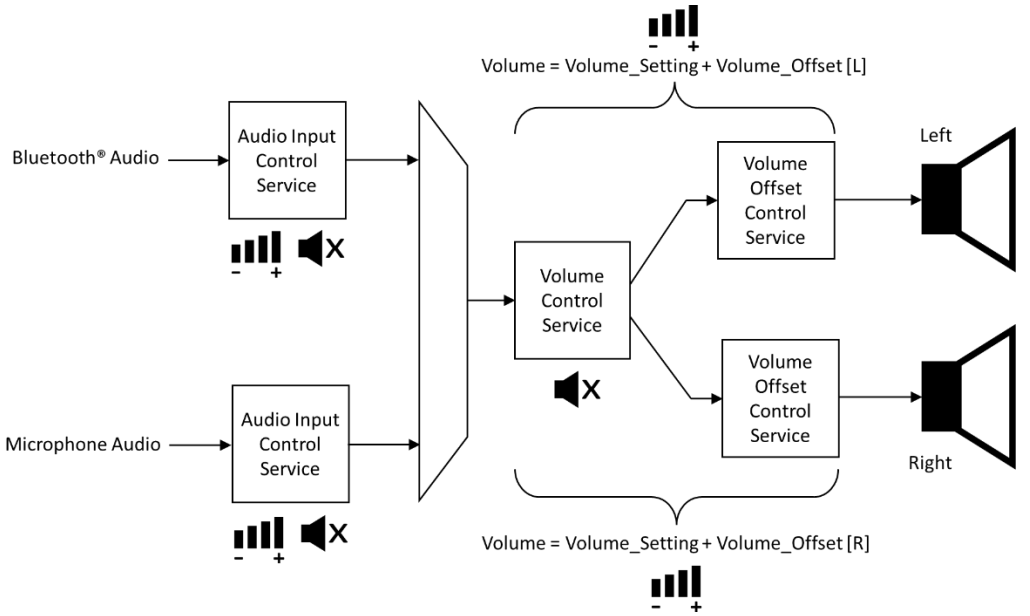


Figure 10.3 Typical implementation for a pair of headphones

Figure 10.4 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 be identical, but with a VOCS instance for the right Audio Location.

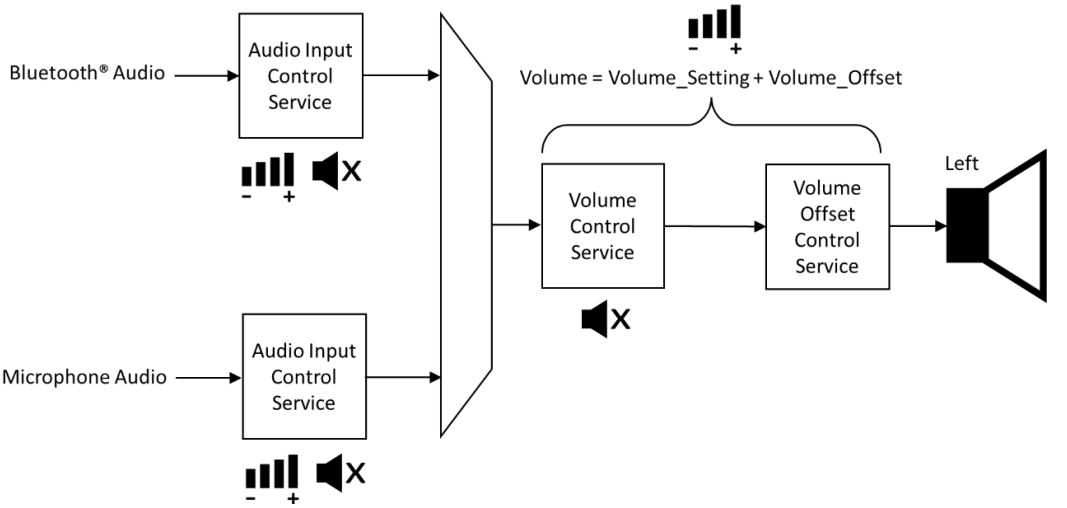


Figure 10.4 Typical implementation for a left hearing aid from a stereo pair

10.6 Microphone control

Although the examples above include microphones, they are all using the microphone as a local input, which 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 device wide control over Microphones.

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.

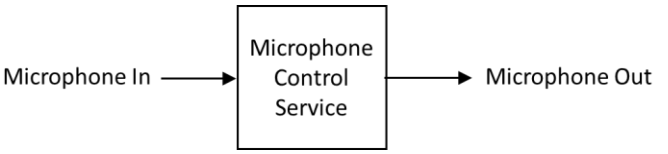


Figure 10.5 Simple usage of the Microphone Control Service

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

Figure 10.6 MICS Mute characteristic values

If control of microphone gain is required, MICS should be combined with an instance of AICS (Figure 10.7), 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 retain MICS.

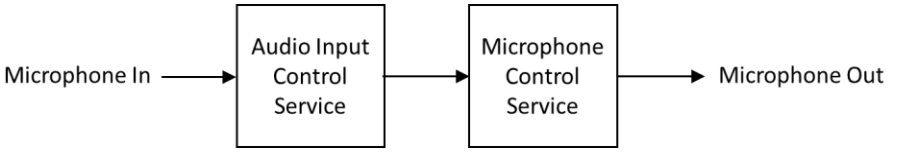


Figure 10.7 Combining the Microphone Service with an instance of the Audio Input Control Service

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 10.8).

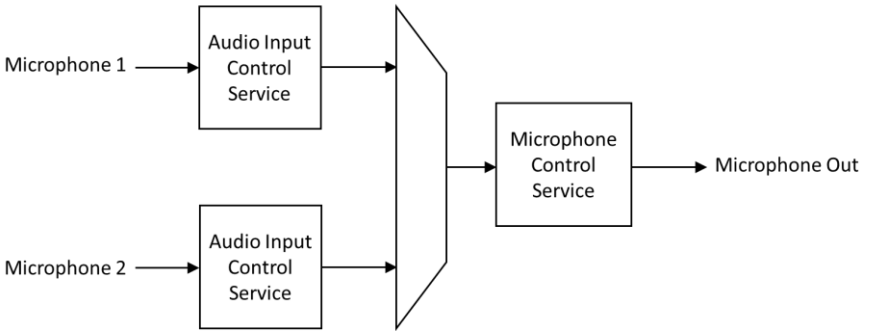


Figure 10.8 Use of MICS and AICS for multiple microphones

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 10.9. 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 10.9 does is demonstrate the flexibility of volume and microphone control services in Bluetooth LE Audio.

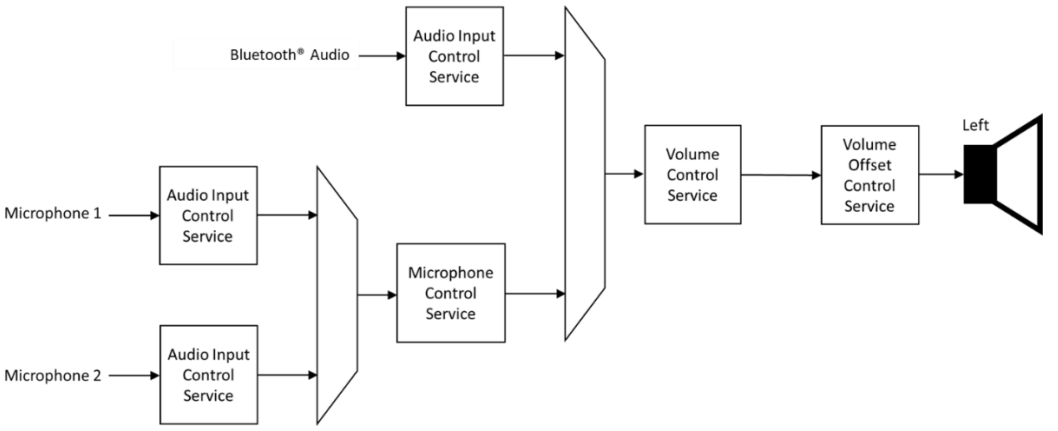


Figure 10.9 The combination of Microphone and Volume Control Services

10.7 A codicil on terminology

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 collocated[48] 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 10.10 illustrates some of the profiles and services described in this chapter in a simple case of an Initiator, Acceptor and independent Commander.

[48] 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".

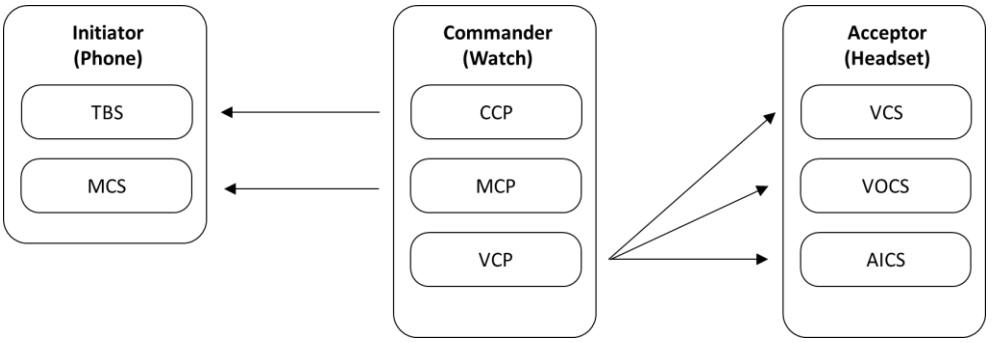


Figure 10.10 An example of Profile and Server relationships for volume and control

In this case, three profiles are implemented in the Commander. Two of them – Call Control 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 colocated; the services in the Acceptor are collocated.

That brings us to the end of the GAF specifications. The only other specifications within Bluetooth LE Audio are the top level profiles, which we are about to come to.

Chapter 11. Top level Bluetooth® LE Audio profiles

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, 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 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 complementary service specifications. HAP – the Hearing Access Profile[49] 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 TMAS service, which is included in the TMAP specification. 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.

A number of top level profiles are currently being developed in the Bluetooth working groups, but the first three scheduled for adoption are:

- HAP and HAS – the Hearing Access Profile and Service, which define requirements for products which are intended for use in the hearing aid ecosystem.
- TMAP – the Telephony and Media Audio Profile. This aims to cover the main features of the classic HFP and A2DP profiles, adding in new capabilities arising from the Bluetooth LE Audio topologies.
- PBP – the Public Broadcast Profile, which is a foundation of the Bluetooth LE Audio Sharing use case (see Chapter 12). It identifies that a broadcast Audio Stream can be received by any Bluetooth LE Audio Acceptor.

These are publicly available in draft form, but are still subject to change. The contents of this chapter reflect their state at the time of writing, which is documented in Section 13.2.2.

[49] 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.

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 buying 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, using the 16_2 LC3 codec settings for voice (16kHz sampling rate, 7 kHz bandwidth) and the 24_2 codec 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 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" through the document. These are:

- A single hearing aid, which renders a single Bluetooth LE Audio Stream,
- A single hearing aid that receives separate left and right Audio Streams and combines the decoded audio into a single Audio Channel,
- 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, and
- A hearing aid that uses a single Bluetooth link but provides separate left and right audio outputs to each 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.

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 a hearing aid. 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).

The bulk of the Hearing Access Profile lists mandatory combinations of BAP and CAP features which are required for different product iterations. Whilst many are obvious, such as setting the CSIS Size characteristic to 2 for a pair of hearing aids, 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 able to receive a Unicast or Broadcast Audio Stream that complies to the BAP mandated requirements, but does not need to receive one which has been encoded with other optional QoS settings. It must accept volume control commands from any device which supports the HARC role, which could be a stand-alone remote control or an application on a phone. For the first time, this means that hearing aid accessories will be globally interoperable with any make of hearing aid.

All hearing aids need to support the «Ringtone», «Conversational» and «Media» Context Types, which mean 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 presets 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 is 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 procedure to other set members based on a notification from one of them. This is the Synchronized Locally operation, which informs a Commander that any 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.

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 11.1.

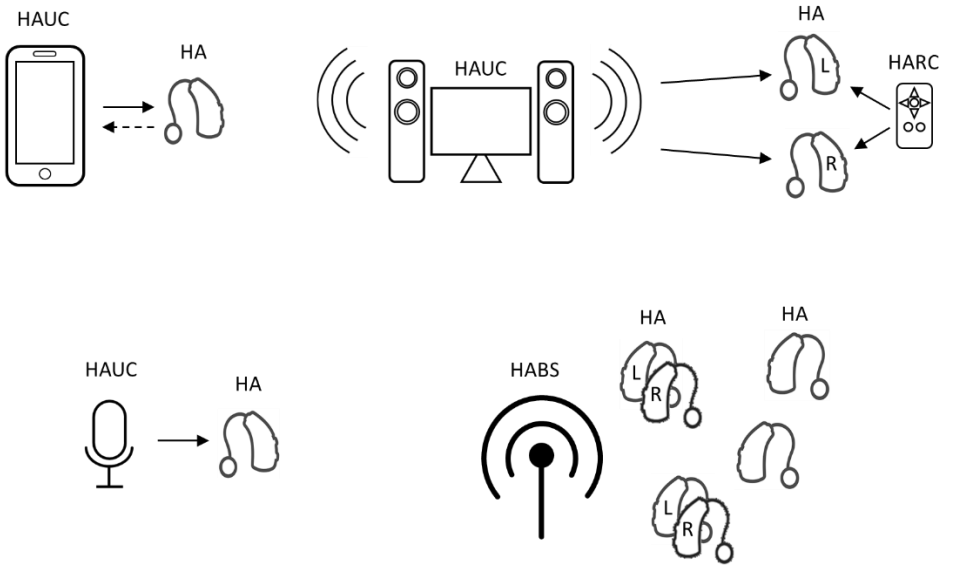


Figure 11.1 The four main use cases for HAP, showing the roles used in each

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.

A final nuance in the Hearing Access Profile is a set of requirements in Section 4.1, which dictate Link Layer setting 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. 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.

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 or characteristics, so incorporates a minimal TMAS section within TMAP.

Because it is bundling both telephony and media applications into one document, TMAP feels like a portmanteau profile, where implementers have a wide-ranging 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 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.

To try and prevent any surprises, the TMAS 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 can also be an extension speaker, or a microphone for a conference phone. Some common configurations are shown in Figure 11.2

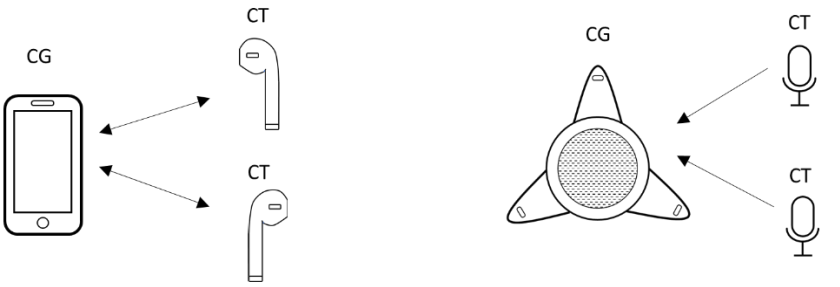


Figure 11.2 Typical configurations for the CG and CT roles

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, but does not mandate any further features above those mandated in CCP.

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 and the Unicast Media Receiver an Acceptor. Typical configurations are shown in Figure 11.3.

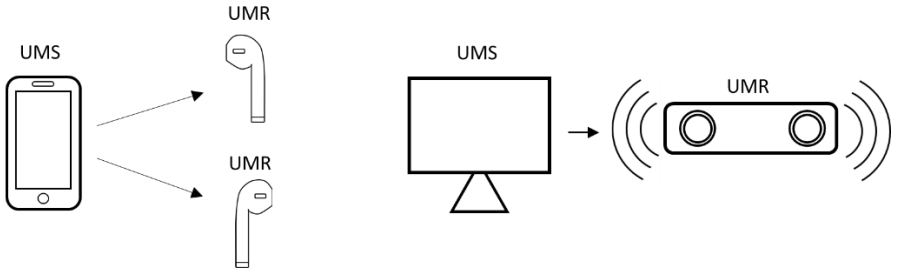


Figure 11.3 Typical TMAP Unicast Media applications

For Unicast Media, TMAP pushes up the audio quality, requiring a Unicast Media Receiver to support all six of the 48kHz sampling codec configurations, form 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 is largely a marketing decision.

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.

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 and Broadcast Media Receiver 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 11.4.

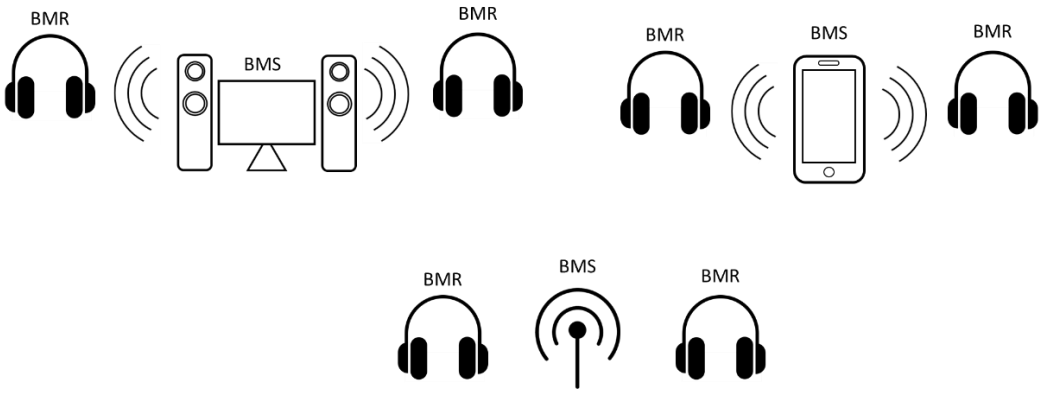


Figure 11.4 Typical use cases for the Broadcast Media Sender and Receiver roles

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.

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.

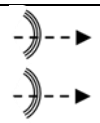
Audio Configuration	Stream Direction (Initiator – Acceptor)	BMS	BMR
12		M	M
13		M	M
14		O	M

Table 11.1 Audio Configuration requirements for Broadcast Media Roles

11.3 Public Broadcast Profile

The Public Broadcast Profile (PBP) is a simple, but very interesting profile. It is intended to support the Audio Sharing use case, providing a guarantee that a broadcast Audio Stream is configured such that any Acceptor can receive it. It contains a new UUID – the Public Broadcast Service UUID, which is added alongside a 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 synchronise to the Periodic Advertising train and then receive and parse the BASE. This reduces the amount of work for the scanner, reducing 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 the Broadcast Sink can decode.

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 may only include the PBP UUID in its Basic Audio Announcement if at least one of the BISes included in the associated BIG has been 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 can only be transmitted when a BIS with these mandatory settings is active.

The reason for PBP is to alert Bluetooth LE Audio Sinks to a broadcast Audio Stream which can be universally received.

That concludes our coverage of the Bluetooth LE Audio specifications. The final chapter will look at some of the new applications they enable.

Chapter 12. Bluetooth® LE Audio applications

The whole point of developing Bluetooth LE Audio was to support new audio applications, not just to produce a slightly lower power alternative to the existing Bluetooth Classic Audio profiles. Part of that was driven by the need to catch up with proprietary extensions, particularly for True Wireless Stereo. The bigger prize was to allow innovation in audio, to keep up the momentum which TWS and voice assistants had generated, making audio more universal and allowing greater flexibility in how we can listen to it.

Broadcast is the child of the telecoil. The telecoil system has been around a long time. 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. 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.

An initial concern 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 hear 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 systems within conference centres with 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. Everyone wanted Bluetooth LE Audio to do it better.

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 and the Commander, providing ways for that information to be sent from a Broadcaster that is trusted or known to an individual listener. As the specifications developed, it quickly became obvious that this provided a very powerful mechanism for users, both to pick up broadcast codes, and gain access to individual broadcasts in a far more flexible manner than we have ever seen with previous Bluetooth connections.

12.1 Changing the way we acquire and consume audio

Changing the way 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.

Commanders mean that we will be able to do far more in terms of selection and control without touching our phones. 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 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. There is a long way to go, but 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.

The ability to use multiple Commanders 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 they 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 parts of 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.

12.2 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 Bluetooth LE Audio by 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 market. 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.

12.2.1 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 12.1, mimicking what is already done today for discovering and pairing to Wi-Fi and Bluetooth devices.

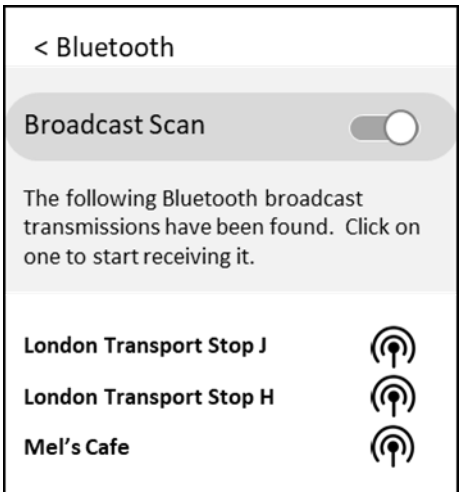


Figure 12.1 Simple user interface to find Bluetooth® LE Audio Broadcasts

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 12.1 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.

There are several tools within the Bluetooth LE Audio specifications which help the development of a good user experience for integrated applications. Looking back at the chapters on broadcast, there are a number of pieces of information which should be included in the broadcast advertisements, to help a scanning device filter and sort different broadcasts. These are:

- The Local Name AD Type, which provides the device name. Normally this would be the primary item of information displayed in a list of available broadcast transmitters.
- The Program_Info LTV. An LTV structure which includes information about the content which is being broadcast. It's similar to the top level of information that is included in a TV's Electronic Program Guide, e.g., the TV channel.
- The BASE, which may contain further information about the content, including the name of the content, e.g., the specific program name for a TV broadcast, as well as the language of transmission.

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.

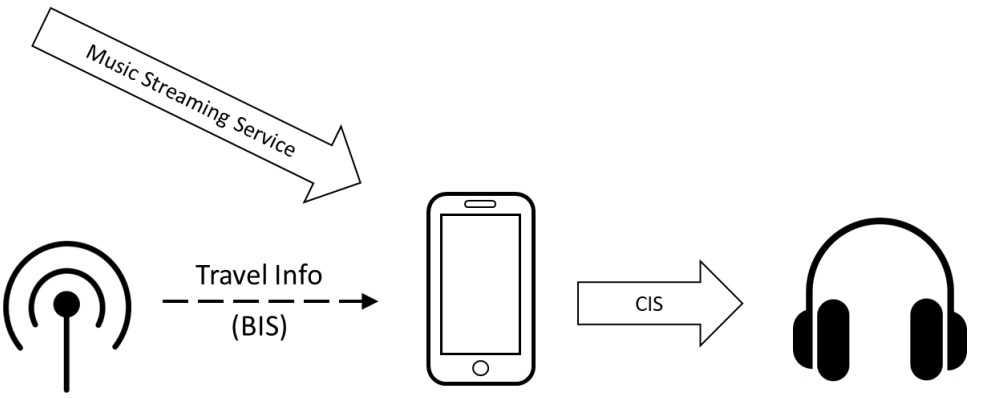


Figure 12.2 Mixing broadcast announcements into an audio stream

Figure 12.2 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 12.3 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.

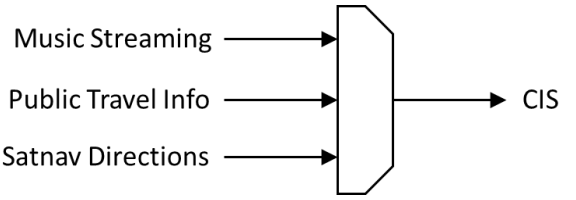


Figure 12.3 Mixing application specific audio streams into a CIS

Given that people are starting to leave their earbuds in for longer, using the transparency mode to hold conservations, this is a useful way of receiving relevant announcements, particularly if they are "silent" foreign language announcements.

12.2.2 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. Directions can tell you which way to turn and where to look. 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.

12.2.3 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 though 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 conservation. 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 within twelve months of the specifications being adopted, devices like this should be readily available for a relatively low price. Product designers should make sure that they can be easily configured by the venue owners, so that they are easy for customers to discover.

12.2.3.1 Making audio universally available

This is where the Public Broadcast Profile becomes important. A commercial Bluetooth LE Audio transmitter should allow the selection of any of the QoS settings which are defined in the BAP specification [BAP Table 6.4]. However, not all Acceptors will be able to decode all of them. Resource constrained devices, such as hearing aids, which want to achieve the greatest possible battery life, will probably not support decoding for sampling rates above 24 kHz. So, if a broadcast transmitter only supports a higher rate, wearers will be unable to hear the broadcast audio. There are some simple solutions to this, but they require broadcast device manufacturers to implement them.

The simplest approach is to confine transmission of music in a public space to the 24_2_1 QoS setting. If there is any background noise, i.e., it is less that a perfect listening environment, most users are likely to find this satisfactory.

Sampling Rate	Mono airtime	Stereo Airtime	Universally receivable?
24 kHz (24_2_1)	13%	26%	Yes
32 kHz (32_2_1)	15%	30%	No
48 kHz (48_2_1)	30%	60%	No
Note: These figures are based on the Low Latency settings. The 48kHz sampling figures have a higher airtime requirement, as the configuration requires a RTN setting of 4, rather than the value of 2 which is used for the 24_2_1 and 32_2_1 settings.			

Table 12.1 Airtime requirements for different broadcast stream QoS settings

Table 12.1 provides the data for this discussion. Given current marketing trends in the audio industry, it's likely that many manufacturers will want to be able to promote their products as "48kHz audio quality", which poses a dilemma. If these transmitters support a stereo 48kHz stream, yet still want to provide universal support, they need to add at least a 24kHz mono stream. If that were included within the same BIG, it would take the airtime to 90%, because although it uses less airtime, every BIS in a BIG is allocated the same timing parameters. The alternative is to transmit a second, separate BIG for the 24kHz mono stream where the combined BIGs would account for around 72% of airtime, but would require more advertising resource. If a broadcast transmitter uses Wi-Fi to obtain its audio stream, that is probably not going to work. A more practical solution, which addresses the quality and universal access requirements would be to transmit 32kHz stereo and 24kHz mono Audio Streams in one BIG, which only consumes around 45% of the total airtime. Transmitting both a stereo 24kHz pair and a 32kHz pair of audio streams in the same BIG takes the same airtime as a single 48kHz stereo stream.

In the examples above, I have chosen the Low Latency QoS settings, despite that fact that they include fewer retransmissions for the 24kHz and 32kHz sampling rates. In this particular application, latency isn't that important, as there are usually no visual cues, or simultaneous ambient sound (although there could be). However, in this application, the broadcast transmitter is likely to be placed high on a wall or attached to the ceiling to get the best coverage, so there's not much in the way to absorb the transmissions. That should mean that the Low Latency settings are adequate. If the café owner located the broadcast transmitter in a cupboard or under the serving counter, that would be different. This is something that equipment designers need to think about in their physical design and installation instructions, ensuring that broadcast transmitters for this application can easily be wall mounted and that any leads attached to a unit are long enough to do that.

Equipment manufacturers need to understand these constraints and design products with the flexibility to support multiple BIGs and allow the installer and venue owner to position and configure them appropriately. We should not expect café owners to understand airtime or QoS parameters – manufacturers should provide them with solutions which are easy to install and use. That includes the ability to customise all of the AD Type names and LTV strings with values which are appropriate for each installation.

A basic broadcast transmitter which just accepts an audio input, either from a 3.5mm jack, RCA plug or a USB input is about as simple as you can get, but will be adequate for very many premises and applications. Equipment vendors will almost certainly provide more complex broadcast units for larger establishments, as well as managed services to configure them, supply the audio stream and integrate them into other audio services. Even with basic Bluetooth LE Audio broadcast, there are plenty of opportunities for differentiation.

12.3 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.

12.3.1 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 being able to hear 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 installations, where the sound would not be heard or be intrusive

These installations are all akin to the simple Broadcaster that we looked at above. They don't need encryption; they just need a broadcast transmitter. That could be a similar 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 location to be entered so that a user can match an audio broadcast with the TV, or it can take the Wi-Fi route of sticking its access information on a notice on the wall.

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 an 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 mono coding, it should be possible to provide six different language streams from one broadcast transmitter. Each input stream would mix the flight announcements in real-time with the audio input for that program, with any unannounced languages being mixed at a subsequent point where they

don't conflict with any ambient announcements. If a transmitter needed to support more channels, it could add a second Bluetooth transmitter. Each language is identified by a Language LTV value in the BASE, indicating the language of each BIS. A user can set their scanning application to select specific languages, so that these are presented preferentially. Alternatively, it can 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 to understand how to use it.

12.3.2 Personal – 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 heard regularly by multiple users, would expect each user to pair and bond with them. 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 the broadcast stream location using PAST, followed by 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 a 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 receive the information they need from the TV to start audio streaming, with no further action from the user. 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 synchronisation or synchronising 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.

If 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.

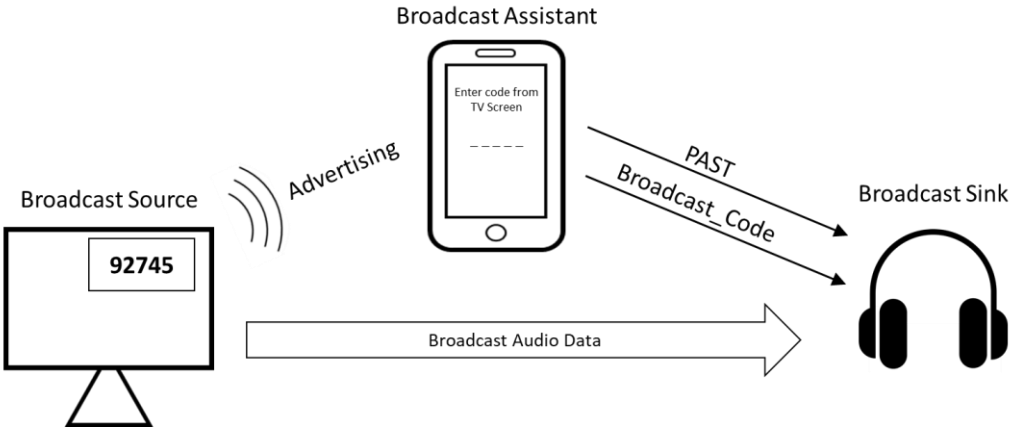


Figure 12.4 An example of how to add a friend to a TV broadcast

The method of obtaining the Broadcast_Code is not described in the Bluetooth LE Audio specifications, as it is out of scope. It could be discovered by displaying a number on the TV screen for a user to enter into a phone app, displaying a QR code, or the use of NFC or another proximity technology. The market will probably agree on a standard option in the near future.

It is expected that Broadcast_Code values would be refreshed at the end of each session, to ensure ongoing privacy. In practice that is likely to happen automatically at each power-down. Bonded users would be unaffected, as they would automatically be issued with the new value each time they connect.

12.3.3 Hotels

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, allowing 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 the Broadcast_Code whenever they want to connect. Equally, the hotel could rely on a user having a general-purpose broadcast scanning app which lets them enter a pin-code from the TV, or scan a QR code, with their phone acting as a Commander for their earbuds. Although these work, they are cumbersome and don't give a particularly smooth user experience. A more effective option

would be to incorporate the Broadcast_Code provision into the hotel app, so that as soon as someone checks in, their phone would be provided with the correct information to access the TV from that app, with their personal Broadcast_Code static for the duration of their stay. That's a fairly simple extension of the TV management systems which are already integrated in most hotel TVs.

12.4 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 original Walkman, you'd see pairs of people sharing their earbuds 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, transition from a unicast stream (or from an A2DP stream – we don't know what underlying technology will be used) 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 Broadcaster, your application will almost certainly want to encrypt your audio, so your phone application needs to distribute the Broadcast_Code. This can be accomplished through a short-term pairing (without a requirement to bond). That has the advantage that your friend's phone will notify yours once it acquires the broadcast stream. Your phone can then stop transmitting the advertising set, so that nobody else is aware of your broadcasts. If other friends want to join in, you just repeat the process. At any point in the process, your friends can leave. Once they have all left, your phone will probably revert back to unicast, as the acknowledged packets in CISes means it is more power efficient for an Initiator – a broadcast transmitter has to transmit every possible retransmission Subevent.

Equally, any of your friends can take the role of Broadcaster to share their favourite music. It's a process which can continue for as long as any of them want. As applications develop, we may find ways to make swapping the Broadcaster within a group even more seamless.

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.

Phones can even 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 could set it up as a private Broadcaster, broadcasting its microphone pickup to everyone else around the table. It uses no cellular functionality, but acts as a local community microphone.

12.5 Audio Sharing

All of these use cases have the same thing in common – they're evolving the telecoil experience, which was a hearing aid only experience, to everybody who owns a Bluetooth LE hearable, whether that's a hearing aid, headphone, set of earbuds, or even a portable speaker. That's a new concept for the vast majority of people. The Bluetooth SIG, working with hearing loss user groups, as well as hearing aid and consumer hearables manufacturers, is developing guidelines for manufacturers and venues to promote the universal nature of broadcast under the name "Audio Sharing" to help encourage its uptake and ensure interoperability for all users.

The interoperability issue is an important one for broadcasts as there are two conflicting categories of devices:

- Hearing aids and resource-constrained hearables, generally supporting the Hearing Access profile with the ability to support only the 16kHz and 24kHz sampling rates for LC3, and
- Consumer devices which may want to use the highest LC3 sampling rates of 48kHz.

To cope with this discrepancy, and ensure the optimum experience for all users, the Audio Sharing guidance separates broadcast transmitters into two categories – Public and Personal.

- Public broadcast transmitters are ones which everyone should be able to access. These must transmit a stream at the lower sampling rates of either 16kHz or 24kHz, supporting the Public Broadcast Profile to inform users where and when those audio streams are available. A user detecting the Public Broadcast Service UUID will know that they can synchronise to that stream. If they have the resources, public devices can also transmit higher quality streams at the same time.
- Personal broadcast transmitters are devices like TVs, phones, PCs and tablets. The Audio Sharing guidelines recognise that the user may normally prefer to receive a higher quality stream, which would not be universally interoperable. However, when a user selects an Audio Sharing compliant broadcast mode on their device, such as when a friend asks to access your TV or listen to your music stream, it must be possible to configure it to a universal setting, which, if selected would broadcast a 16kHz or 24kHz sampled, PBP compliant Audio Stream, so that any device could pick it up. It is up to the user to ask the other stream recipients which option they need. Again, if the device is capable, it can simultaneously transmit streams with higher sampling rates.

The Bluetooth SIG plans to promote these guidelines in a similar manner to the way that the Wi-Fi Alliance has for publicly accessible Wi-Fi access points, to help encourage interoperability and confidence in the new Audio Sharing ecosystem.

12.6 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.

The techniques to find the Broadcast Source and to obtain the Broadcast_Code are the same as the ones described above, but these particular use cases are generating interest in developing an industry-wide method to connect.

12.6.1 Tap 2 Hear – making it simple

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.

12.6.2 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. 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.

12.6.3 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.

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.

In designing Bluetooth LE Audio Controllers, it is a useful exercise to think about how you would implement them on a battery box. It's such an easy thing for users to interact with, but because of its inherent simplicity, it's often overlooked. It won't necessarily be the best device to implement them on, but its accessibility and small size, fitting into pockets that won't hold a phone, make it an attractive alternative.

As designers find ways to turn small devices like this into compelling user interfaces, we will probably find that we spend less time interacting with our phone. As earbuds get better at mixing Bluetooth audio streams with ambient sound and conversations (which audio processing can enhance), we will probably spend more time wearing our earbuds, talking to all sorts of things around us and interacting with audio. Phones are not going to go away any time soon, although we have already passed peak smartphone, but Bluetooth LE Audio may drive the applications that lead us to whatever comes next.

12.7 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 Audio Sharing 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.

12.7.1 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.

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.

12.7.2 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 in a piconet 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 Delegators and the use of the characteristics in BASS are fundamental to the sharing and authentication processes described above.

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.

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 ecosystem, where a user can pick and choose multiple products

from different manufacturers and have confidence that they will all work with each other.

12.7.3 Balancing quality and application

Developers should not forget the difference between Optional and Mandatory in the Bluetooth LE Audio specifications. In many of them, very little is mandated. Many features are mandated to be supported, such as the mandated codec settings in BAP and TMAP, but that doesn't mean that an application needs to use them. 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 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, which should be designed with that in mind. The audio quality is there when it is needed, but so is a lot else.

12.7.4 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 tail of the dog. 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 the hearables' tail could start wagging the dog. 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 13. 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.

13.1 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
BFI	Bad Frame Indication
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
BW	Bandwidth
CAP	Common Audio Profile
CAS	Common Audio Service

Acronym / Initialism	Meaning
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
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)
HAP	Hearing Access Profile
HARC	Hearing Aid Remote Controller
HAS	Hearing Access Service
HAUC	Hearing Aid Unicast Client
HCI	Host Controller Interface

Acronym / Initialism	Meaning
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)
NESN	Next Expected Sequence Number
NPI	Null Payload Indicator
NSE	Number of Subevents
OOB	Out of Band
OTP	Object Transfer Profile

Acronym / Initialism	Meaning
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
TMAS	Telephony and Media Audio Service
TNS	Temporal Noise Shaping
UCI	Uniform Caller Identifier
UI	User Interface
uint48	unsigned 48-bit integer

Acronym / Initialism	Meaning
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 13.1 Acronyms and initialisms

13.2 Bluetooth LE Audio specifications

The following specifications are part of the new specifications developed for Bluetooth LE Audio. They build on new features which are present in the Core version 5.2 and above.

13.2.1 Adopted Specifications

These specifications have been adopted and implementations can be qualified to them

Specification	Full Name	Family
AICS	Audio Input Control Service	Generic Audio Framework
ASCS	Audio Stream Control Service	Generic Audio Framework
BAP	Basic Audio Profile	Generic Audio Framework
BASS	Broadcast Audio Scan Service	Generic Audio Framework
CCP	Call Control Profile	Generic Audio Framework
CSIP	Coordinated Set Identification Profile	Generic Audio Framework
CSIS	Coordinated Set Identification Service	Generic Audio Framework
GMCS	Generic Media Control Service (part of MCS)	Generic Audio Framework
GTBS	Generic Telephone Bearer Service (part of TBS)	Generic Audio Framework
LC3	Low Complexity Communication Codec	Codec
MCP	Media Control Profile	Generic Audio Framework
MCS	Media Control Service	Generic Audio Framework
MICP	Microphone Control Profile	Generic Audio Framework
MICS	Microphone Control Service	Generic Audio Framework

Specification	Full Name	Family
PACS	Published Audio Capabilities Service	Generic Audio Framework
TBS	Telephone Bearer Service	Generic Audio Framework

Table 13.2 Adopted Bluetooth® LE Audio Specifications

13.2.2 Draft specifications

These specification are undergoing interoperability testing. Draft versions have been publicly released to help implementers understand the contents, but are subject to change. The text in this edition is based on the version shown in Table 13.3.

Specification	Version	Full Name	Family
CAP	VS_r06	Common Audio Profile	GAF
CAS	VS_r07	Common Audio Service	GAF
HAP	v09	Hearing Access Profile	Top level profile
HAS	v09	Hearing Access Service	Top level profile
PBP	VS_r00	Public Broadcast Profile	Top level profile
TMAP	VSr02	Telephony and Media Audio Profile	Top level profile
TMAS		Telephony and Media Audio Service (part of TMAP)	GAF

Table 13.3 Draft Bluetooth® LE Audio Specifications

13.3 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 Audio Contexts discovery	BAP	5.4
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

Procedure or sub-procedure name	Specification	Section
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
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

Procedure or sub-procedure name	Specification	Section
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
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

Procedure or sub-procedure name	Specification	Section
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
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

Procedure or sub-procedure name	Specification	Section
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 13.4 Procedures and sub-procedures defined in Bluetooth® LE Audio specifications

13.4 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 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
Broadcast Audio Scan Control Point	BASS	3.1
Broadcast Receive State	BASS	3.1
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

Characteristic	Specification	Table
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
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
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
Volume Control Point	VCS	3.1

Characteristic	Specification	Table
Volume Flags	VCS	3.1
Volume Offset Control Point	VOCS	3.1
Volume Offset State	VOCS	3.1
Volume State	VCS	3.1

Table 13.5 Characteristics defined in the Bluetooth® LE Audio specifications

13.5 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
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
Call Control Client	CCP	Section 2
Call Control Server	CCP	Section 2
Call Gateway (CG)	TMAP	Section 2.2

Phrase	Specification	Section
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 4.5.4.1.1
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
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

Phrase	Specification	Section
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

Table 13.6 Defined terms in the Bluetooth® LE Audio specifications

Index

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 low energy. 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 Low Energy and LE Audio. Since 2013 he has chaired the Bluetooth Hearing Aid working group, which developed the concept of Bluetooth LE Audio and he 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 hearable 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 hearables.