ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη ενσωματωμένης εφαρμογής πάνω σε Bluetooth Low Energy development board και εφαρμογής smart-phone με σκοπό τη δημιουργία ενός συστήματος καταγραφής καιρού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευστάθιος Ι. Παπατζανάκης

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη ενσωματωμένης εφαρμογής πάνω σε Bluetooth Low Energy development board και εφαρμογής smart-phone με σκοπό τη δημιουργία ενός συστήματος καταγραφής καιρού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευστάθιος Ι. Παπατζανάκης

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 25η Ιουλίου 2014.

.....	.....	.....
Κιαμάλ Πεκμεστζή	Δημήτριος Σούντρης	Γεώργιος Οικονομάκος
Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2014

.....
Ευστάθιος Ι. Παπατζανάκης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Αντικείμενο της παρούσας διπλωματικής εργασίας είναι η μελέτη της στοίβας πρωτοκόλλων του Bluetooth Low Energy (BLE), ή Bluetooth Smart, όπως είναι η εμπορική του ονομασία, με σκοπό τη δημιουργία ενός συστήματος καταγραφής καιρού (σταθμός καιρού). Το σύστημα αποτελείται από δύο τμήματα. Το πρώτο τμήμα είναι αυτό που εκτελείται σε ενσωματωμένο σύστημα και καταγράφει δεδομένα για τον καιρό, χρησιμοποιώντας μια σειρά από κατάλληλους αισθητήρες (π.χ. θερμοκρασία, υγρασία, ατμοσφαιρική πίεση). Το δεύτερο τμήμα είναι αυτό που εκτελείται σε κάποιο smartphone ή tablet, το οποίο λαμβάνει τα δεδομένα από το ενσωματωμένο σύστημα, τα επεξεργάζεται, και παρουσιάζει στον χρήστη της εφαρμογής μία διεπαφή χρήστη (user interface), η οποία του επιτρέπει να βλέπει συγκεντρωτικά στοιχεία για τον καιρό, καθώς και να καθορίζει τις διάφορες παραμέτρους λειτουργίας του συστήματος. Η επικοινωνία μεταξύ των δύο τμημάτων του συστήματος γίνεται μέσω του πρωτοκόλλου BLE. Το BLE έχει ως βασικό χαρακτηριστικό τη μικρή κατανάλωση ενέργειας, κάτι που επιτρέπει στο ενσωματωμένο σύστημα να λειτουργεί με μπαταρία για μεγάλο χρονικό διάστημα.

Το ενσωματωμένο σύστημα που χρησιμοποιήθηκε είναι το DA14580 της εταιρίας Dialog Semiconductor. Το DA14580 είναι ένα μικρού μεγέθους BLE System on a Chip (SoC), το οποίο περιέχει όλα τα απαραίτητα στοιχεία για τη δημιουργία μιας ενσωματωμένης εφαρμογής, η οποία θα εκτελείται πάνω στο σύστημα, χρησιμοποιώντας μπαταρία και λίγα εξωτερικά στοιχεία. Συγκεκριμένα, χρησιμοποιήσαμε την αναπτυξιακή πλακέτα που παρέχεται από την εταιρία, με σκοπό τη δημιουργία και τον έλεγχο εφαρμογών πριν αυτές περάσουν στην παραγωγή. Για το δεύτερο τμήμα του συστήματος, υλοποιήθηκε μια εφαρμογή (app) για το λειτουργικό σύστημα Android, η οποία μπορεί να εκτελεστεί σε τελευταίας τεχνολογίας smartphones ή tablets, που υποστηρίζουν το πρωτόκολλο BLE.

Λέξεις – Κλειδιά : Bluetooth, Bluetooth Low Energy, BLE, Bluetooth Smart, Android, Smart Phone, Tablet, Android Application, Android App, System on a Chip, SoC, Embedded System, Sensor, Weather Station, Temperature Sensor, Humidity Sensor, Atmospheric Pressure Sensor, UV Index, DA14580, Dialog Semiconductor, Development Board

Abstract

The purpose of the present diploma thesis is the study of Bluetooth Low Energy (BLE) protocol stack, which is commercially known as Bluetooth Smart, with the objective of creating a weather recording system (weather station). The system consists of two parts. The first part is the one that runs on an embedded system and uses appropriate sensors (e.g. temperature, humidity, atmospheric pressure) to record weather related data. The second part is the one that runs on a smartphone or tablet, which receives the recorded data from the embedded system, and, after some processing, it presents the user with a user interface, that allows him to view cumulative data about the weather, and set the system's operating parameters and options. The communication between the two parts of the system is achieved through the use of BLE. BLE's most important feature is the low power consumption, which allows the embedded system to operate on battery power for a long period of time.

The embedded system that we used is the DA14580, which is a product of Dialog Semiconductor. DA14580 is a small form factor BLE System on a Chip (SoC), which contains all the necessary elements for the creation of an embedded application, which will run on the embedded system on battery power using only a small number of external components. In particular, we used the development board, which Dialog Semiconductor provides to developers of BLE applications, in order to create and debug their applications before production. For the second part of the system, we created an application (app) for the Android operating system, which may run on last generations of smartphones or tablets, with the requirement that they support BLE.

Key –Words : Bluetooth, Bluetooth Low Energy, BLE, Bluetooth Smart, Android, Smart Phone, Tablet, Android Application, Android App, System on a Chip, SoC, Embedded System, Sensor, Weather Station, Temperature Sensor, Humidity Sensor, Atmospheric Pressure Sensor, UV Index, DA14580, Dialog Semiconductor, Development Board

Ευχαριστίες

Για την εκπόνηση της παρούσας διπλωματικής εργασίας, θα ήθελα να ευχαριστήσω τον καθηγητή κ. Κιαμάλ Πεκμεστζή, για την αμέριστη συμπαράσταση που επέδειξε, καθώς και τον υποψήφιο διδάκτορα Νίκο Ευταξιόπουλο – Σαρρή για την πολύτιμη βοήθεια του. Επίσης θα ήθελα να ευχαριστήσω τους εκπροσώπους του ελληνικού τμήματος της εταιρίας Dialog Semiconductor για την παροχή της αναπτυξιακής πλακέτας και του απαραίτητου υλικού για τη δημιουργία του συστήματος καταγραφής καιρού, καθώς και για τη βοήθεια τους σε ερωτήματα και προβλήματα σχετικά με το αναπτυξιακό, τα οποία προέκυψαν κατά τη διάρκεια της εκπόνησης της εργασίας.

Πίνακας Περιεχομένων

Κεφάλαιο 1

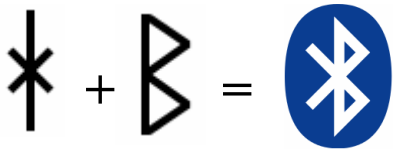
Εισαγωγή

1.1 Εισαγωγικά – Ιστορικά στοιχεία για το Bluetooth

Bluetooth® Logo

Το Bluetooth είναι μια ασύρματη τεχνολογία για τη μεταφορά δεδομένων σε μικρές αποστάσεις. Με τη βοήθεια του Bluetooth, ένας χρήστης μπορεί να συνδέσει διάφορες κινητές ή σταθερές συσκευές του, υλοποιώντας μια σειρά από περιπτώσεις χρήσης, πάνω από το επονομαζόμενο και Personal Area Network (PAN). Για παράδειγμα, οι συσκευές αυτές μπορεί να είναι συσκευές εισόδου (Human Interface Device) για υπολογιστή, όπως ποντίκια και πληκτρολόγια, mp3 players για ασύρματη σύνδεση με ηχεία και αναπαραγωγή μουσικής, κινητά τηλέφωνα για μεταφορά δεδομένων (π.χ. επαφών) μεταξύ τους ή σύνδεση με κάποιο περιφερειακό ακουστικό (headset, hands-free). Η τελευταία περίπτωση χρήσης μπορεί να θεωρηθεί ως ένας από τους βασικότερους λόγους της δημοφιλίας του Bluetooth. Βρίσκεται σε ευρεία χρήση ακόμη και σήμερα, παρότι άλλες περιπτώσεις χρήσης, όπως η μεταφορά δεδομένων μεταξύ κινητών, δε χρησιμοποιούνται, ή ακόμα και δεν υποστηρίζονται πλέον, όσο παλαιότερα. Όπως θα δούμε παρακάτω, με την έλευση του Bluetooth Low Energy, η γκάμα των συσκευών που μπορούν να περιλαμβάνονται στο PAN του χρήστη, εμπλουτίζεται με μια ολοένα και αυξανόμενη ποικιλία διαφορετικών προϊόντων.

Η προέλευση του ονόματος του Bluetooth, που άρχισε να χρησιμοποιείται επισήμως το 1998, αντικατοπτρίζει ακριβώς τη δυνατότητα, που παρέχει η τεχνολογία αυτή στον χρήστη, να συνδέει με ευκολία τις διάφορες συσκευές του, καθώς και την ενοποίηση των πολλών διαφορετικών προτύπων επικοινωνίας σε ένα. Το όνομα προέρχεται από απόδοση στα αγγλικά του επιθέτου ενός Δανού βασιλιά του 10^{ου} αιώνα, ο οποίος κατάφερε να ενώσει τις διάφορες δανικές φυλές, που ήταν ως τότε διαχωρισμένες, σε ένα ενιαίο βασίλειο. Το όνομα του βασιλιά αυτού ήταν Harald Bluetooth (Blåtand/Blåtann στα σκανδιναβικά). Επίσης το επίσημο logo του Bluetooth, που φαίνεται στην παραπάνω εικόνα, προέρχεται από τη συνένωση των γραμμάτων του ρουνικού αλφάβητου, που αποτελούν τα αρχικά του Harald Bluetooth.



Το Bluetooth αναπτύχθηκε, αρχικά, από τους μηχανικούς της εταιρίας προϊόντων τηλεπικοινωνίας Ericsson, το 1994, σε μια προσπάθειά τους να απαλλαγούν από τη χρήση των σειριακών καλωδίων RS-232, που χρησιμοποιούνταν ευρέως εκείνη την εποχή. Το 1998, η Ericsson μαζί με 4 άλλες εταιρίες (Intel, IBM, Nokia, Toshiba) δημιουργούν μια εμπορική ένωση με το όνομα Bluetooth SIG (Special Interest Group), με σκοπό την εξέλιξη και προώθηση της νέας τεχνολογίας. Μέχρι το τέλος του έτους το Bluetooth SIG απαριθμούσε ήδη πάνω από 400 μέλη, ενώ σήμερα αποτελείται από περισσότερα από 20000 μέλη. Το SIG επιβλέπει την εξέλιξη της τεχνολογίας και των προτύπων του Bluetooth. Επίσης ελέγχει τη χρήση της τεχνολογίας και των εμπορικών σημάτων από τους διάφορους κατασκευαστές συσκευών, μέσω ειδικών τεστ τα οποία πρέπει να περάσουν οι συσκευές πριν βγουν στην κυκλοφορία, προκειμένου να θεωρηθούν συμβατές με το πρότυπο και να μπορούν να χρησιμοποιήσουν το λογότυπο του Bluetooth. Κατά καιρούς διοργανώνει, επίσης, τα επονομαζόμενα UnPlugFest, στα οποία μηχανικοί από εταιρίες μέλη του SIG μπορούν να συναντηθούν και να ανταλλάξουν ιδέες σχετικά με την τεχνολογία, ενώ παράλληλα πραγματοποιούν τεστ, στα οποία πολλές παλιές και νέες συσκευές Bluetooth ενεργοποιούνται

ταυτόχρονα στον ίδιο χώρο, προκειμένου να ελεγχθεί η δυνατότητα interoperability υπό δύσκολες συνθήκες, κάτι που αποτελεί ένα από χαρακτηριστικά προώθησης της συγκεκριμένης τεχνολογίας.

Το 1999 βγαίνει η πρώτη έκδοση του Bluetooth specification (1.0), ενώ το ενδιαφέρον για τη νέα αυτή τεχνολογία αυξάνεται. Την επόμενη χρονιά εμφανίζονται στην αγορά τα πρώτα προϊόντα που χρησιμοποιούν το Bluetooth για ασύρματη επικοινωνία, όπως το πρώτο κινητό τηλέφωνο με υποστήριξη Bluetooth καθώς και το πρώτο Bluetooth ακουστικό. Από εκεί και πέρα, η τεχνολογία γίνεται ολοένα και πιο δημοφιλής και τα προϊόντα που τη χρησιμοποιούν αυξάνονται με πολύ γρήγορους ρυθμούς, για να φτάσουμε στο 2005 να έχουμε πάνω από 5 εκατομμύρια Bluetooth chipsets να εισέρχονται στην αγορά κάθε εβδομάδα, και το 2008 να υπάρχουν σχεδόν 2 δισεκατομμύρια προϊόντα με υποστήριξη του πρωτοκόλλου.

Εν τω μεταξύ, το 2004, βγαίνει η έκδοση 2.0 του Bluetooth, η οποία περιλαμβάνει ως βασικό νέο χαρακτηριστικό το Enhanced Data Rate (EDR), το οποίο, όπως λέει και το όνομά του, παρέχει υψηλότερη ταχύτητα από την προηγούμενη έκδοση, αναφερόμενη από εκεί και πέρα ως Basic Rate (BR). Στα μέσα του 2007 βγαίνει η έκδοση 2.1, βασικό χαρακτηριστικό της οποίας ήταν το Secure Simple Pairing (SSP), το οποίο απλοποιεί σε μεγάλο βαθμό τη διαδικασία ασφαλούς σύνδεσης (pairing ή bonding) μεταξύ δύο συσκευών, αυξάνοντας παράλληλα το παρεχόμενο επίπεδο ασφάλειας. Το 2009 έχουμε την έκδοση 3.0, με βασικό χαρακτηριστικό το Alternate MAC/PHY (Media Access Control / Physical Layer), το οποίο, χρησιμοποιώντας ένα διαφορετικό μέσο μεταφοράς στο χαμηλότερο επίπεδο (το specification ορίζει το 802.11 MAC/PHY), μπορεί να επιτύχει ακόμα μεγαλύτερες ταχύτητες μεταφοράς δεδομένων.

Έτσι, φτάνουμε πλέον στο 2010, οπότε με την έκδοση 4.0 του Bluetooth, εμφανίζεται μια νέα τεχνολογία με το όνομα Bluetooth Low Energy (BLE). Από εδώ και πέρα, η παλαιότερη έκδοση του Bluetooth αναφέρεται ως Bluetooth Classic ή Bluetooth BR/EDR. Επειδή η διπλωματική εργασία ασχολείται με το BLE και όχι με το Bluetooth Classic, η εισαγωγική παρουσίαση του BLE γίνεται παρακάτω, αφού αναφέρουμε κάποια βασικά στοιχεία για το Bluetooth Classic.

1.2 Bluetooth Classic – Bluetooth BR/ EDR

Παρότι έχουν το ίδιο όνομα και προέρχονται από τον ίδιο οργανισμό, τα Bluetooth Classic και Bluetooth Low Energy, ενώ έχουν κάποιες ομοιότητες σε ορισμένα σημεία τους, εμφανίζουν μεγάλες διαφορές σε άλλα. Αυτό είναι λογικό αν σκεφτεί κανείς ότι δημιουργήθηκαν για να εξυπηρετήσουν διαφορετικούς σκοπούς. Ακολουθούν, επιγραμματικά, τα βασικά χαρακτηριστικά του Bluetooth Classic. Κάποια από αυτά υπάρχουν και στο BLE, όπου χρησιμοποιούνται με διαφορετικό, συνήθως απλούστερο, τρόπο. Αυτή είναι και μια βασική διαφορά μεταξύ των δύο τεχνολογιών. Το Bluetooth Classic είναι ένα αρκετά πολύπλοκο πρωτόκολλο. Το BLE αντίθετα κατασκευάστηκε εξαρχής και με γνώμονα την απλότητα των διαδικασιών και των πρωτοκόλλων.

1.2.1 Radio

Το Bluetooth χρησιμοποιεί το επονομαζόμενο ISM (industrial, scientific and medical) εύρος συχνοτήτων, το οποίο εκτείνεται από τα 2.4 GHz μέχρι τα 2.4835 GHz (τμήμα του Ultra high frequency – UHF band). Το συγκεκριμένο εύρος συχνοτήτων είναι ελεύθερο προς χρήση διεθνώς, οπότε δε χρειάζονται ειδικές άδειες για τη χρήση του. Αυτό, βέβαια, το κάνει αρκετά δημοφιλές και για άλλες τεχνολογίες, όπως το WiFi, καθώς και για πολλές συσκευές που στέλνουν δεδομένα ασύρματα χρησιμοποιώντας ιδιωτικά πρωτόκολλα. Αυτό έχει ως αποτέλεσμα ότι μπορεί να υπάρχει αρκετή συμφόρηση στις συχνότητες αυτές, κάτι που καθιστά δύσκολη τη μεταφορά δεδομένων. Το Bluetooth, με βάση το specification του, λαμβάνει υπόψη του το συγκεκριμένο περιορισμό, και προσπαθεί να αποφύγει τη συμφόρηφη χρησιμοποιώντας το Adaptive Frequency Hopping. Η ίδια τεχνική ακολουθείται και στο BLE, οπότε θα την περιγράψουμε με λεπτομέρεια στο αντίστοιχο κεφάλαιο.

Το παραπάνω εύρος συχνοτήτων χωρίζεται σε ένα σύνολο από 79 κανάλια, καθένα από τα οποία έχει εύρος 1 MHz γύρω από μια κεντρική συχνότητα, που ξενικά από τα 2402MHz.

$$f_c = 2402MHz + k, k = 0,1,...,78$$

Όπως βλέπουμε, υπάρχουν αχρησιμοποίητες συχνότητες και στα δύο άκρα του φάσματος. Αυτό είναι έτσι από σχεδιασμού, προκειμένου να αποφεύγονται πιθανές παρεμβολές που μπορεί να υπάρχουν στα άκρα του φάσματος από τεχνολογίες που χρησιμοποιούν τα γειτονικά του ISM φάσματα, καθώς επίσης, και πιθανές παρεμβολές του ίδιου του Bluetooth στα γειτονικά φάσματα.

Το symbol rate του Bluetooth Classic είναι 1 Ms/s. Το BR χρησιμοποιεί Gaussian Frequency Shift Keying (GFSK) modulation με ένα bit ανά σύμβολο και ταχύτητα 1Mbps. Το EDR χρησιμοποιεί διαφορετικό modulation, το Phase Shift Keying (PSK), σε δύο εκδοχές: π/4-DQPSK και 8DPSK. Αυτά παρέχουν υψηλότερο symbol rate, οπότε οι αντίστοιχες ταχύτητες είναι 2Mbps και 3Mbps.

Ορίζονται 3 κλάσεις συσκευών με βάση τη μέγιστη ισχύ εκπομπής.

Power Class	**Range**	**Maximum Output Power**	**Nominal Output Power**	**Minimum Output Power**
1	~1m	100mW (20dBm)	N/A	1mW (0dBm)
2	~10m	2.5mW (4dBm)	1mW (0dBm)	0.25mW (-6dBm)
3	~100m	1mW (0dBm)	N/A	N/A

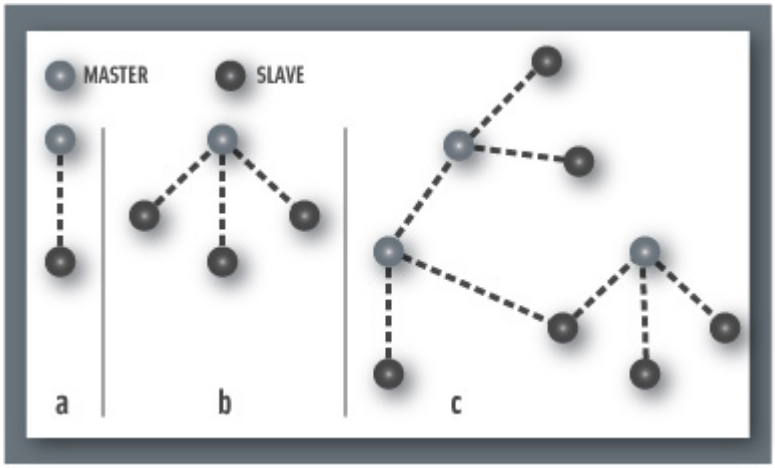
Στον παραπάνω πίνακα ως Minimum Output Power ορίζεται η ελάχιστη ισχύς εκπομπής στη μέγιστη ρύθμιση ισχύος της συσκευής. Οι συσκευές κλάσης 2 και 3 δεν απαιτούν κάποιου είδους ελέγχου της ισχύος εκπομπής, κάτι όμως που απαιτείται για τις συσκευές κλάσης 1, οι οποίες μεταδίδουν σήμα μεγαλύτερο των 4dBm.

Στην πλευρά του δέκτη ορίζεται ως επίπεδο ευαισθησίας λήψης μικρότερο ή ίσο των -70dBm για bit error rate (BER) ίσο με 0.1%. Το ίδιο ακριβώς ισχύει και στο BLE. Το παραπάνω σημαίνει ότι ένας δέκτης θα πρέπει να μπορεί να λαμβάνει σήμα -70dBm (100pW) και να το αποκωδικοποιεί με μέγιστη ανοχή το ένα λάθος ανά χίλια bit.

1.2.2 Baseband – Link Manager – Host Cotroller Interface

Το Bluetooth είναι ένα πρωτόκολλο βασισμένο στην αποστολή πακέτων με μια αρχιτεκτονική master-slave. Ο master είναι η κεντρική συσκευή επικοινωνίας σε μια ομάδα που αποτελείται από τον ίδιο και μέχρι 7 slaves, οι οποίοι μπορούν να επικοινωνούν μόνο με το master και όχι μεταξύ τους. Περισσότεροι των 7 slaves μπορούν να υπάρχουν, αλλά αυτοί πρέπει να βρίσκονται σε κατάσταση parked, κατά την οποία δεν μπορούν να στείλουν δεδομένα. Οι ρόλοι μπορούν να αλλάξουν κατά τη διάρκεια της σύνδεσης, αλλά πάντα υπάρχει μόνο ένας master. Ο master παρέχει το κεντρικό ρολόι και όλοι οι slaves είναι συγχρονισμένοι με αυτό. Το ρολόι αυτό χτυπάει με ρυθμό ένα χτύπο ανά 312,5μs. Δύο χτύποι του ρολογιού αποτελούν ένα slot (625μs), ενώ δύο slots αποτελούν ένα slot pair (1,25ms). Ο master στέλνει δεδομένα σε μονά slots, ενώ ένας slave σε ζυγά slots. Το ανάποδο ισχύει για τη λήψη. Κάθε πακέτο μπορεί να καταλαμβάνει 1, 3 ή 5 slots.

Η ομάδα αυτή του master και των μέχρι 7 slaves ονομάζεται piconet. Μία συσκευή μπορεί να συμμετέχει ως master ή slave σε περισσότερα του ενός piconets, σχηματίζοντας έτσι μια δομή που ονομάζεται scatternet, όπως φαίνεται στην ακόλουθη εικόνα. Στις περιπτώσεις (a) και (b) έχουμε piconet με ένα ή περισσότερους slaves, ενώ στην περίπτωση (c) έχουμε πολλά piconets που ορίζουν ένα scatternet. Το specification του Bluetooth, όμως, δεν ορίζει κάποιο τρόπο επικοινωνίας μεταξύ συσκευών που βρίσκονται μεν στο ίδιο scatternet, αλλά σε διαφορετικά piconets.



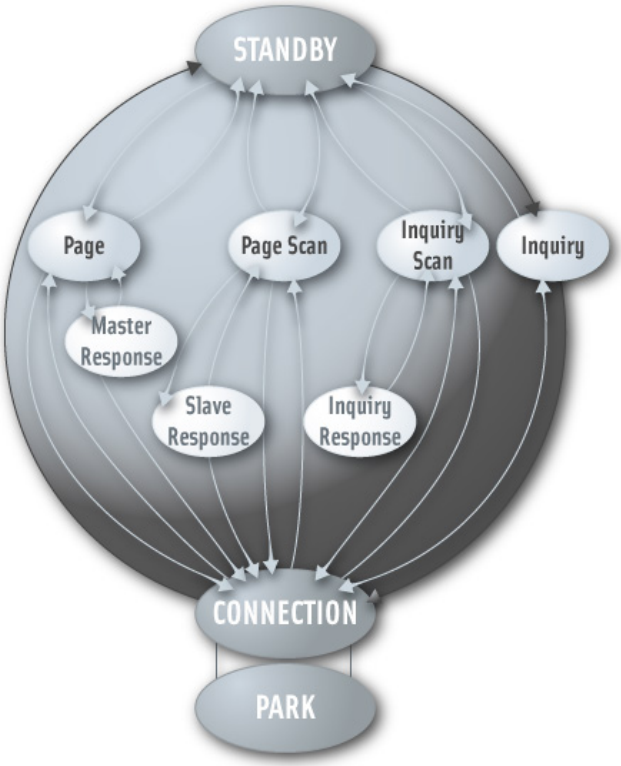
Η βασική μορφή του πακέτου φαίνεται στην ακόλουθη εικόνα. Κάθε πακέτο ξεκινά με ένα Access Code, το οποίο καθορίζει το είδος του πακέτου και τους δέκτες στους οποίους απευθύνεται.

Στην περίπτωση του EDR, και, για λόγους προς τα πίσω συμβατότητας, χρησιμοποιείται μια διαφορετική μορφή πακέτου, στην οποία το modulation στο φυσικό επίπεδο αλλάζει κατά τη διάρκεια της αποστολής. Αυτό εξασφαλίζει ότι συσκευές που δεν υποστηρίζουν το EDR, μπορούν να αναγνωρίσουν το αρχικό κομμάτι του πακέτου, το οποίο μεταδίδεται στο GFSK του BR, και να σταματήσουν τη λήψη πριν την αλλαγή του modulation στο FSK του EDR.

Το Baseband του Bluetooth είναι το τμήμα εκείνο του συστήματος που ελέγχει την πρόσβαση στο φυσικό επίπεδο. Το baseband ορίζει μια σειρά από κανάλια στο φυσικό επίπεδο (physical channels), τα οποία χρησιμοποιούνται για διαφορετικές λειτουργίες του πρωτοκόλλου.

- **Inquiry Scan Physical Channel:** Χρησιμοποιείται κατά τη διαδικασία εύρεσης συσκευών (device discovery). Μια συσκευή (slave) που είναι σε discoverable mode πραγματοποιεί τη διαδικασία inquiry scan, ελέγχοντας το κανάλι για τυχόν αποστολή inquiry πακέτων από μια συσκευή (master) που πραγματοποιεί αναζήτηση. Αυτό είναι το αντίθετο από ό,τι στο BLE, όπου η discoverable συσκευή είναι αυτή που στέλνει τα κατάλληλα πακέτα.
- **Page Scan Physical Channel**: Χρησιμοποιείται κατά τη διαδικασία σύνδεσης μεταξύ δύο συσκευών. Η συσκευή που στέλνει τα πακέτα paging γίνεται master μετά τη σύνδεση, ενώ η συσκευή που πραγματοποιεί τη διαδικασία page scan γίνεται slave.
- **Basic/Adapted Piconet Physical Channel:** Χρησιμοποιούνται κατά τη σύνδεση συσκευών σε ένα piconet για την ανταλλαγή δεδομένων.

Ακριβώς πάνω από το φυσικό επίπεδο, και μέσα στο baseband, υπάρχει επίσης ο Link Controller, ο οποίος ελέγχει τις διαδικασίες σύνδεσης στο χαμηλότερο επίπεδο, με βάση τις ανταλλαγές πακέτων που αναφέρθηκαν παραπάνω. Το state-machine του Link Controller φαίνεται στην παρακάτω εικόνα.

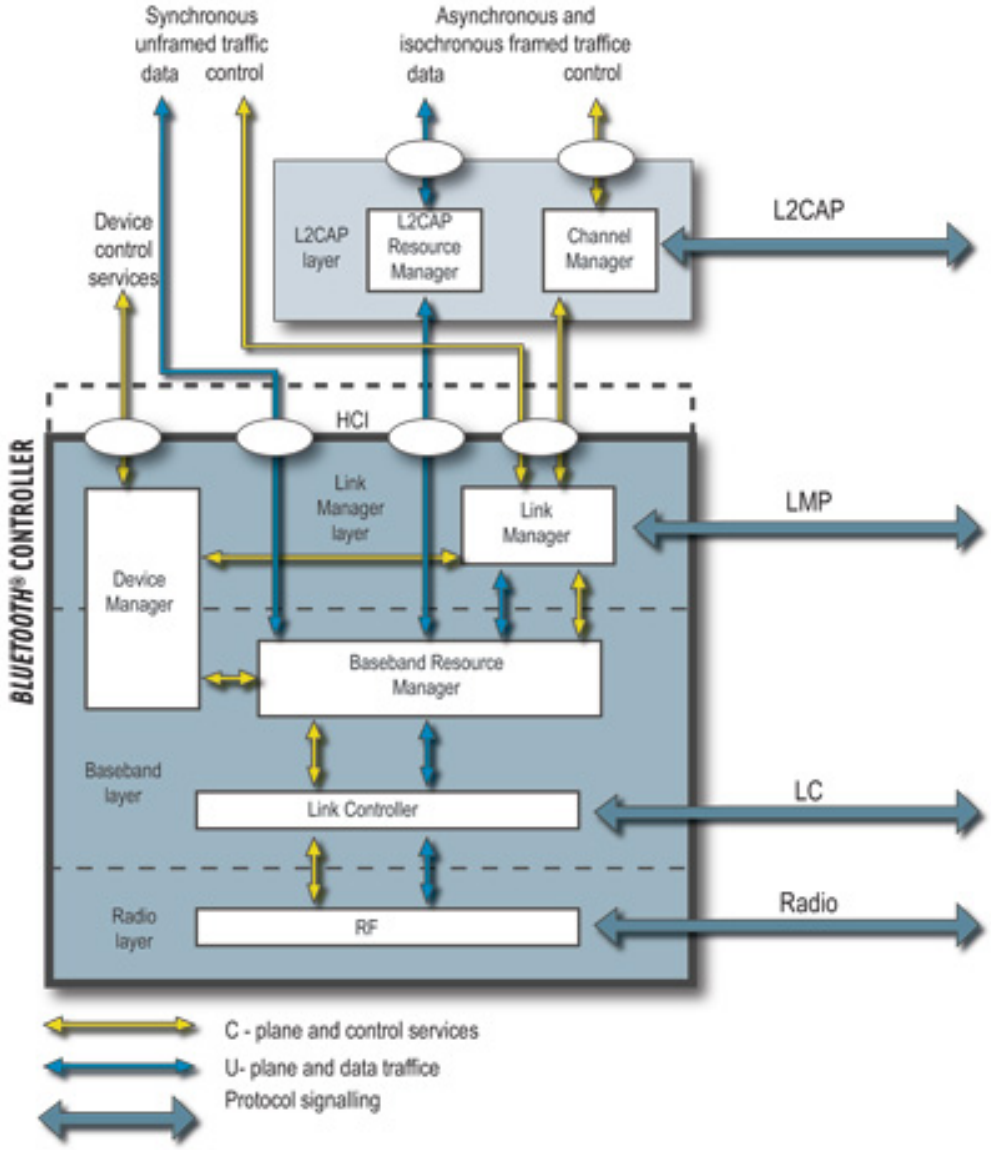


Πάνω στα physical channels ορίζονται physical links. Κάθε physical link είναι συσχετισμένο με ακριβώς ένα physical channel, παρέχοντας επιπλέον δυνατότητες, όπως ρύθμιση ισχύος και κρυπτογράφηση. Πάνω από τα physical links ορίζονται τα logical transports, με σημαντικότερα τα Synchronous Connection-Oriented (SCO), Extended Synchronous Connection-Oriented (eSCO) και Asynchronous Connection-Oriented (ACL). Τέλος, πάνω από αυτά ορίζονται logical links, όπως τα Link Control (LC) και ACL Control (ACL-C), που χρησιμοποιούνται για τον έλεγχο της σύνδεσης, καθώς και τα User Asynchronous/Isochronous (ACL-U), User Synchronous (SCO-S) και User Extended

Synchronous (eSCO-S), που χρησιμοποιούνται για ασύγχρονη ή σύγχρονη μεταφορά δεδομένων, ανάλογα με την εκάστοτε περίπτωση χρήσης.

Σε αυτό το σημείο έχουμε περάσει πλέον στο επίπεδο ζεύξης δεδομένων, όπου υπάρχει ο Link Manager, ο οποίος ελέγχει όλα τα παραπάνω logical channels και links. Οι Link Manager διαφορετικών συσκευών επικοινωνούν μεταξύ τους χρησιμοποιώντας το Link Manager Protocol. Μέσω αυτού του πρωτοκόλλου μπορούν να εκτελέσουν διαδικασίες όπως σύνδεση, αποσύνδεση, έλεγχο ισχύος, καθορισμό Quality of Service (QoS), ενεργοποίηση του EDR, αλλαγή ρόλων, καθώς και διαδικασίες σχετικές με την ασφάλεια, όπως authentication, pairing (π.χ. με χρήση Secure Simple Pairing), κρυπτογράφηση.

Το Baseband και ο Link Manager αποτελούν τον Controller του συστήματος, ενώ τα ανώτερα επίπεδα αποτελούν τον Host. Μεταξύ των δύο υπάρχει το επονομαζόμενο Host Controller Interface (HCI), το οποίο ορίζει ένα σύνολο από εντολές ελέγχου των στοιχείων του controller και γεγονότων που σχετίζονται με τη λειτουργία του. Το HCI μπορεί να είναι εσωτερικό στο Bluetooth chipset ή να υλοποιείται μέσω ενός εξωτερικού transport layer, όπως USB ή UART. Η όλη αρχιτεκτονική του συστήματος φαίνεται στην ακόλουθη εικόνα.



1.2.3 Τα ανώτερα επίπεδα

Ακριβώς πάνω από το HCI, έχουμε το Logical Link Control and Adaptation Protocol (L2CAP). Αυτό παρέχει υπηρεσίες πολύπλεξης στα πρωτόκολλα των ανώτερων επίπεδων, καθώς και τη δυνατότητα διάσπασης και ανασύνθεσης μεγάλων πακέτων δεδομένων. Η λειτουργία του L2CAP βασίζεται στην έννοια του καναλιού. Κάθε κανάλι έχει ένα αναγνωριστικό (channel identifier – CID), και μπορεί να είναι connectionless ή connection-oriented. Το L2CAP χρησιμοποιείται, σε πολύ απλούστερη μορφή, και από το BLE.

Στα ανώτερα επίπεδα του host, έχουμε πρωτόκολλα μεταφοράς δεδομένων μεταξύ peers, όπως το RFCOMM, το πρωτόκολλο εύρεσης παρεχόμενων υπηρεσιών (Service Discovery Protocol – SDP), καθώς και το Generic Access Profile (GAP) που καθορίζει τις διαδικασίες με τις οποίες δυο συσκευές μπορούν να ανακαλύψουν η μία την άλλη και να συνδεθούν. Στο GAP θα αναφερθούμε εκτενέστερα στην περιγραφή του σε σχέση με το BLE.

Τέλος, στο επίπεδο εφαρμογής έχουμε τα Profiles, τα οποία είναι specifications για συγκεκριμένες περιπτώσεις χρήσης. Το SIG έχει ορίσει ως πρότυπα μια σειρά από profiles για τις συνηθισμένες χρήσεις του Bluetooth. Η έννοια του profile υπάρχει και στο BLE, όπου και πάλι καθορίζει συγκεκριμένες περιπτώσεις χρήσης . Όμως, ο ορισμός τους στο BLE γίνεται με διαφορετικό, απλούστερο και ενιαίο τρόπο, όπως θα δούμε στο αντίστοιχο κεφάλαιο. Κάποια από τα σημαντικότερα Bluetooth Profiles είναι τα ακόλουθα:

- **A2DP – Advanced Audio Distribution Profile:** Περιγράφει πώς μουσική ποιότητας στέρεο μπορεί να μεταφερθεί από μία συσκευή (π.χ. ένα mp3 player) σε μια άλλη (π.χ. ένα σύστημα αναπαραγωγής μουσικής).
- **HFP – Hands-Free Profile:** Χρησιμοποιείται για τη σύνδεση μεταξύ μιας Bluetooth hands-free συσκευής (ακουστικού) και ενός κινητού τηλεφώνου.
- **HSP – Headset Profile:** Χρησιμοποιείται για την ασύρματη σύνδεση μεταξύ Bluetooth ακουστικών και μιας συσκευής αναπαραγωγής ήχου.
- **HID – Human Interface Device Profile:** Χρησιμοποιείται για την ασύρματη σύνδεση μεταξύ συσκευών εισόδου, όπως ποντίκια και πληκτρολόγια, με υπολογιστές.
- **GOEP – Generic Object Profile:** Χρησιμοποιείται για τη μεταφορά αντικειμένων μεταξύ συσκευών.
- **SYNC – Synchronization Profile:** Χρησιμοποιείται σε συνδυασμό με το GOEP για τον συγχρονισμό δεδομένων ημερολογίου και επαφών μεταξύ συσκευών.

1.3 Bluetooth Low Energy (BLE)

1.3.1 Ιστορικά στοιχεία

Η τεχνολογία του Bluetooth Low Energy πρωτοεμφανίστηκε το 2006 με την ονομασία Wibree από την εταιρία παραγωγής προϊόντων τηλεπικοινωνίας Nokia. Οι ερευνητές της Nokia, ήδη από το 2001, είχαν ξεκινήσει μια προσπάθεια να καθορίσουν περιπτώσεις χρήσης, υπαρκτές και νέες, για τις οποίες οι υπάρχουσες τεχνολογίες ασύρματης μετάδοσης δεν επαρκούσαν ή ήταν δύσχρηστες. Η προσπάθεια αυτή κατέληξε το 2004 σε ένα project με την ονομασία Bluetooth Low End Extension, το οποίο χρησιμοποιούσε στοιχεία από το πρότυπο του Bluetooth, με τελικό σκοπό τη δημιουργία μιας ασύρματης τεχνολογίας με τη μικρότερη δυνατή κατανάλωση ενέργειας. Αποτέλεσμα όλων αυτών ήταν το Wibree. Λιγότερο από ένα χρόνο μετά την κυκλοφορία του, όμως, η Nokia, η οποία είναι και ιδρυτικό μέλος του Bluetooth SIG, αποφάσισε να δώσει την τεχνολογία του Wibree στο SIG, προκειμένου αυτή να περιληφθεί στο μελλοντικό πρότυπο του Bluetooth, χρησιμοποιώντας το ήδη γνωστό εμπορικό σήμα και υπό τον έλεγχο του SIG. Έτσι, το 2010, με την κυκλοφορία της έκδοσης 4.0 του Bluetooth, το Bluetooth Low Energy εμφανίστηκε ως μία πολύ χαμηλής κατανάλωσης και χαμηλού κόστους ασύρματη τεχνολογία, με την εμπορική ονομασία Bluetooth Smart.

Εκτός από το Bluetooth Smart, υπήρχαν ήδη στην αγορά και ανταγωνιστικές αντίστοιχες τεχνολογίες, όπως το ANT ή το Zigbee. Ασχέτως των δυνατοτήτων και περιορισμών της κάθε τεχνολογίας, το Bluetooth Smart έχει το μεγάλο πλεονέκτημα ότι υποστηρίζεται από ένα οργανισμό όπως το SIG, ο οποίος έχει μεγάλο αριθμό μελών, και στον οποίο συμμετέχουν όλοι οι μεγάλοι κατασκευαστές software και hardware, ενώ χρησιμοποιεί το ήδη ευρέως διαδεδομένο και δημοφιλές Bluetooth brand name.

Ένα σημαντικό γεγονός για το Bluetooth Smart ήταν η υποστήριξή του από το iPhone 4S, που κυκλοφόρησε τον Οκτώβριο του 2011, το πρώτο smartphone με υποστήριξη της έκδοσης 4.0 του Bluetooth. Έτσι, άρχισαν να εμφανίζονται στην αγορά και τα πρώτα Bluetooth Smart gadgets, ενώ όλο και περισσότερα smartphones υποστήριζαν το νέο πρωτόκολλο. Στο λειτουργικό σύστημα Android, η επίσημη υποστήριξη ξεκίνησε σχετικά πρόσφατα με την έκδοση 4.3, ενώ κάποιοι κατασκευαστές παρείχαν ήδη υποστήριξη με δικές τους βιβλιοθήκες. Με την κυκλοφορία των Windows 8, η Microsoft παρέχει επίσης native υποστήριξη για τη νέα τεχνολογία, ενώ, πλέον, πολλοί σταθεροί ή φορητοί υπολογιστές πωλούνται με ενσωματωμένο Bluetooth 4.0 hardware. Οι προοπτικές για την εξέλιξη του Bluetooth Smart είναι πολύ ευοίωνες και αναλυτές εκτιμούν ότι το πλήθος των συσκευών με υποστήριξη του προτύπου αναμένεται να εκτοξευθεί τα επόμενα χρόνια.

Όπως είδαμε, η κυκλοφορία του Bluetooth Smart συνέπεσε με την άνθιση της αγοράς των smartphones και tablets, ενώ καινούρια προϊόντα μπορούσαν πλέον να δημιουργηθούν, με βάση τη νέα τεχνολογία, τα οποία δεν ήταν πριν εφικτά. Έτσι, εμφανίζονται έννοιες όπως φορετές συσκευές (wearables), ή appcessory, ένας νεολογισμός που προκύπτει από το συνδυασμό των λέξεων app, που αναφέρεται σε μια εφαρμογή που τρέχει σε κάποιο smartphone ή tablet, και accessory. Ουσιαστικά, πρόκειται για μικρού μεγέθους συσκευές, οι οποίες, λόγω χαμηλής κατανάλωσης, μπορούν να λειτουργούν με μια μπαταρία ακόμα και για χρόνια. Οι συσκευές αυτές, από μόνες τους, δεν έχουν μεγάλη χρησιμότητα (π.χ. δεν έχουν user interface), όμως, σε συνδυασμό με μια εφαρμογή σε κάποιο smartphone, αποκτούν χρηστικότητα που δε θα ήταν αλλιώς δυνατή. Μάλιστα, πολλές διαφορετικές συσκευές μπορούν να συνδυαστούν σε νέες περιπτώσεις χρήσης, δίνοντας έτσι ακόμα μεγαλύτερη χρηστικότητα.

Στην αγορά υπάρχουν ήδη πολλές τέτοιες συσκευές τύπου wearable, που κυρίως έχουν να κάνουν με τη μέτρηση της δραστηριότητας του χρήστη (καρδιακός ρυθμός, αριθμός βημάτων), ενώ, τελευταία, φαίνονται να αποκτούν δημοτικότητα προϊόντα τύπου smart-watch.

Fitbit Flex

Nike+ FuelBand

Οι δυνατότητες βέβαια δε σταματούν εδώ. Υπάρχουν ήδη εφαρμογές στον τομέα της υγείας, όπως θερμόμετρα, πιεσόμετρα και μετρητές γλυκόζης με Bluetooth Smart. Επίσης, η δυνατότητα αποστολής εντολών προς τις συσκευές, αλλά και από τις συσκευές, ανοίγει νέες προοπτικές σε τομείς όπως το έξυπνο σπίτι και το Internet of Things. Με τη βοήθεια του BLE, μπορούμε να έχουμε πολλές τέτοιες συσκευές, μικρού μεγέθους και κόστους και με ελάχιστη κατανάλωση ενέργειας, οι οποίες, συνδυαζόμενες και ελεγχόμενες από κάποιο κεντρικό σημείο, παρέχουν στον χρήστη δυνατότητες ελέγχου, τοπικά ή απομακρυσμένα, και, κυρίως, αυτοματισμού ενεργειών. Για παράδειγμα, ένας αισθητήρας φωτός θα μπορούσε να συνδυαστεί με ένα ηλεκτρικό κύκλωμα στα παράθυρα ώστε να ανοιγοκλείνει να πατζούρια, ανάλογα με την ένταση του ηλιακού φωτός. Και αυτό θα γίνεται χωρίς να χρειάζονται επιπλέον καλώδια και συνδέσεις.

Μια άλλη περίπτωση χρήσης, που αναμένεται να γίνει πολύ δημοφιλής στο μέλλον, και η οποία, παρά την απλότητά της, παρέχει απεριόριστες δυνατότητες στους developers και τους χρήστες είναι ο έλεγχος εγγύτητας (proximity). Υπάρχει η απλή περίπτωση χρήσης για την αποφυγή απώλειας ή την εύρεση αντικειμένων. Για παράδειγμα, ένα Bluetooth Smart μπρελόκ μπορεί να χρησιμοποιηθεί, ώστε ο χρήστης να εντοπίζει τα κλειδιά του ή για να μην τα χάσει ή ακόμα και για να μην ξεχάσει κάπου το τηλέφωνό του, το οποίο θα τον ειδοποιήσει όταν απομακρυνθεί από αυτό. Μια άλλη περίπτωση, που μπορεί ακόμα και να αλλάξει την εμπειρία χρήσης κάποιων υπηρεσιών, είναι η χρησιμοποίηση μεγάλου αριθμού συσκευών Bluetooth Smart και η τοποθέτησή τους σε κάποιο χώρο. Οι συσκευές αυτές, το μόνο που κάνουν, είναι να μεταδίδουν, ανά τακτά χρονικά διαστήματα, μία μικρή ποσότητα δεδομένων, παρέχοντας τη δυνατότητα σε μια κατάλληλη εφαρμογή smartphone να τις αναγνωρίσει. Με αυτό τον τρόπο μπορούμε να έχουμε τεχνικές GPS εσωτερικού χώρου, καθώς και παρουσίαση διαφορετικού περιεχομένου στον χρήστη, ανάλογα με την εγγύτητά του σε συγκεκριμένες συσκευές. Για παράδειγμα, ένας χρήστης εισέρχεται σε ένα μουσείο, αφού έχει κατεβάσει την εφαρμογή που παρέχεται από το μουσείο στο app-store της πλατφόρμας του κινητού του. Το μουσείο είναι γεμάτο από μικρές Bluetooth Smart συσκευές, τις οποίες η εφαρμογή αναγνωρίζει. Έτσι, με την είσοδο του χρήστη στο μουσείο, η εφαρμογή επιβεβαιώνει την κράτηση που είχε κάνει νωρίτερα, χωρίς αυτός να χρειάζεται να περιμένει στην ουρά. Αργότερα, κατά την περιήγησή του, ο χρήστης μπορεί να βλέπει αυτόματα στο κινητό του πληροφορίες για τα εκθέματα που βρίσκονται γύρω του. Παράδειγμα πραγματικής συσκευής τέτοιου τύπου είναι το iBeacon της Apple.

1.3.2 Σχεδιαστικές επιλογές

Το Bluetooth Low Energy σχεδιάστηκε εξαρχής με γνώμονα τη μείωση της κατανάλωσης ενέργειας. Ένας άλλος παράγοντας που ελήφθη υπόψη ήταν το χαμηλό κόστος, αφού στόχος της τεχνολογίας ήταν να χρησιμοποιηθεί σε φτηνά προϊόντα μαζικής παραγωγής. Αν το κόστος ήταν απαγορευτικό για τον τελικό χρήστη, η χρησιμότητα της τεχνολογίας θα ήταν περιορισμένη. Το βασικό όμως χαρακτηριστικό και ο βασικός στόχος ήταν η ελάχιστη δυνατή κατανάλωση ενέργειας. Εδώ πρέπει να αναφέρουμε ότι και το Bluetooth Classic θεωρούνταν τεχνολογία χαμηλής κατανάλωση ενέργειας, όμως, με το BLE, αυτό το χαρακτηριστικό πέρασε σε άλλο επίπεδο. Εκεί, δηλαδή, που μια συσκευή Bluetooth Classic μπορεί να λειτουργεί με μπαταρία για μέρες ή εβδομάδες, μια συσκευή BLE μπορεί να λειτουργεί με μικρότερη μπαταρία για μήνες ή χρόνια.

Αυτή η σχεδιαστική επιλογή είναι εμφανής σε όλα τα σημεία του προτύπου και σε κάθε στρώμα της στοίβας πρωτοκόλλων, από το φυσικό επίπεδο ως το επίπεδο εφαρμογής. Κατά την περιγραφή των τμημάτων του BLE στα επόμενα κεφάλαια, θα δούμε πολλές περιπτώσεις, όπου συγκεκριμένα χαρακτηριστικά και λειτουργίες επελέγησαν, ακριβώς λόγω της συμβολής τους στην εξοικονόμηση ενέργειας. Όπως και στην περίπτωση του Bluetooth Classic, το SIG όρισε στο πρότυπο ολόκληρη τη στοίβα πρωτοκόλλων που πρέπει να υλοποιείται από τις συμβατές συσκευές. Η πολιτική αυτή είναι διαφορετική από αυτή, για παράδειγμα, του WiFi ή του Ethernet, που ορίζουν μόνο συγκεκριμένα στρώματα (ζεύξης δεδομένων και φυσικό). Με τον τρόπο αυτό το SIG έχει μεγαλύτερο έλεγχο πάνω στις συμβατές συσκευές, κάτι το οποίο βοηθά στο interoperability, ενώ, στην περίπτωση του BLE, σημαίνει ότι όλα τα στρώματα του πρωτοκόλλου μπορούν να βελτιστοποιηθούν ως προς την κατανάλωση ενέργειας. Δε θα αναφέρουμε εδώ συγκεκριμένες σχεδιαστικές επιλογές, αφού αυτές περιγράφονται με λεπτομέρεια στα επόμενα κεφάλαια. Αναφέρουμε όμως κάποια γενικά χαρακτηριστικά.

Το πρώτο από αυτά είναι η απλότητα. Σε αντίθεση με το Bluetooth Classic, που είναι ένα αρκετά πολύπλοκο πρωτόκολλο επικοινωνίας, το BLE έχει απλοποιήσει πολλές από τις διαδικασίες. Η απλότητα αυτή σημαίνει χαμηλή κατανάλωση ενέργειας, αφού απαιτείται εκτέλεση λιγότερων εντολών από το micro-processor της συσκευής, αλλά και χαμηλότερο κόστος, αφού η δημιουργία του απαραίτητου firmware είναι ευκολότερη και το ίδιο το firmware μικρότερο σε μέγεθος, απαιτώντας λιγότερη μνήμη.

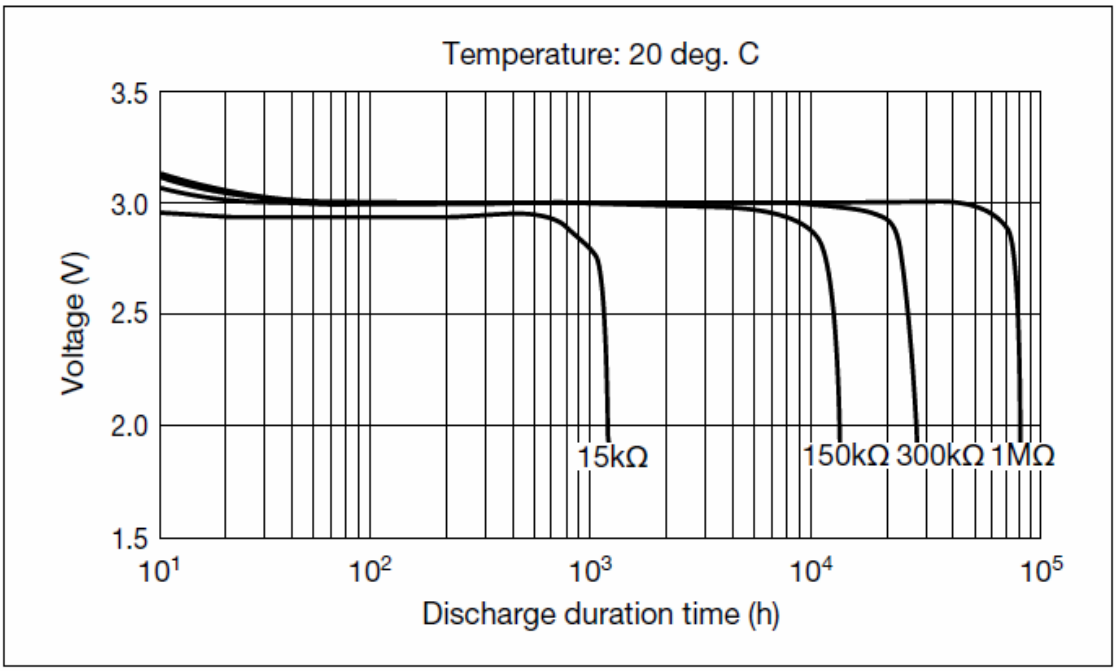
Εφόσον μιλάμε για ασύρματη τεχνολογία, πρέπει να λάβουμε υπόψη ότι το πιο ενεργοβόρο κομμάτι του συστήματος είναι το ίδιο που του επιτρέπει να επικοινωνεί, δηλαδή το κύκλωμα αποστολής και λήψης δεδομένων στο φυσικό επίπεδο. Όταν το ράδιο είναι ανοιχτό και στέλνει ή ελέγχει το φάσμα για απεσταλμένα πακέτα, τότε έχουμε τη μεγαλύτερη κατανάλωση ενέργειας. Αυτό σημαίνει ότι το ράδιο θα πρέπει να παραμένει κλειστό για όσο το δυνατό μεγαλύτερο διάστημα, κάτι το οποίο έχει καθορίσει σε μεγάλο βαθμό τον τρόπο λειτουργίας του BLE.

Ένα άλλο γενικό χαρακτηριστικό του BLE, που προκύπτει επίσης από την ανάγκη μείωσης της κατανάλωσης ενέργειας, είναι η εγγενής ασυμμετρία που παρουσιάζει σε ό,τι αφορά τους peers μιας σύνδεσης. Όπως θα δούμε παρακάτω, αυτοί ονομάζονται master και slave, όπως και στο Bluetooth Classic. Πολλές από τις διαδικασίες που απαιτούνται από το πρωτόκολλο εκτελούνται στον master. Η λογική είναι ότι ο master είναι συνήθως μια συσκευή τύπου smartphone ή tablet, η οποία έχει μεγάλη υπολογιστική ισχύ και λειτουργεί με επαναφορτιζόμενη μπαταρία υψηλής χωρητικότητας. Αντίθετα, ο slave, είναι συνήθως μια συσκευή μικρού μεγέθους και υπολογιστικής ισχύος, η οποία λειτουργεί με μια τυπική μπαταρία του εμπορίου.

Τέλος, αξίζει να αναφέρουμε ότι ως κριτήριο για την κατανάλωση ενέργειας του BLE, επελέγη η δυνατότητα λειτουργίας ακόμα και για χρόνια με μπαταρίες τύπου coin-cell, όπως η CR2032, που απεικονίζεται στην παρακάτω φωτογραφία.

Η τυπική χωρητικότητα των μπαταριών αυτών είναι 200 έως 240 mAh, όμως, στην πράξη, η συνολική ενέργεια που αποδίδουν εξαρτάται από πολλούς παράγοντες και είναι μικρότερη των τιμών αυτών. Επίσης, οι μπαταρίες αυτές έχουν κάποιο μέγιστο ρεύμα, που μπορούν να αποδώσουν σε κάποιο κύκλωμα, χωρίς να υποστούν ζημιά. Αυτό είναι της τάξης των 15 έως 20 mA. Ακόμα και αν δεν προσεγγίζονται αυτές οι τιμές, η συνεχής ζήτηση μεγάλου ρεύματος από την μπαταρία μειώνει τη διάρκεια ζωής της. Οι μπαταρίες αυτές είναι σχεδιασμένες για να παρέχουν μικρές ποσότητες ρεύματος. Οπότε, στο BLE, δεν αρκεί μόνο η μείωση της συνολικής κατανάλωσης, αλλά απαιτείται και η ελαχιστοποίηση του μέγιστου ρεύματος.

Ακολουθεί ένα διάγραμμα, όπου φαίνεται η διάρκεια ζωής μιας μπαταρίας σε σχέση με το φορτίο, καθώς και ένας πίνακας, όπου υπολογίζεται η διάρκεια ζωής μιας μπαταρίας χωρητικότητας 200 mAh σε σχέση με τη μέση κατανάλωση.



Μέση κατανάλωση (μA)	Διάρκεια Ζωής (μέρες)
5	1666
10	833
15	555
20	416
25	333
30	277

Μία μέση κατανάλωση 25 μA αντιστοιχεί, για παράδειγμα, σε ένα duty cycle 5 ms ανά δευτερόλεπτο με μέση κατανάλωση 5 mA.

1.3.3 Περιορισμοί του Bluetooth Low Energy

Όπως συμβαίνει και σε κάθε είδους τεχνολογία που σχεδιάζεται γύρω από ένα συγκεκριμένο χαρακτηριστικό, έτσι και στην περίπτωση του BLE, έγιναν αναγκαστικές υποχωρήσεις σε ορισμένους τομείς. Έτσι, το BLE έχει κάποιους περιορισμούς, που είναι απαραίτητο να γνωρίζει ένας developer, προκειμένου να ελέγξει αν η συγκεκριμένη τεχνολογία είναι κατάλληλη για την εφαρμογή που προτίθεται να δημιουργήσει.

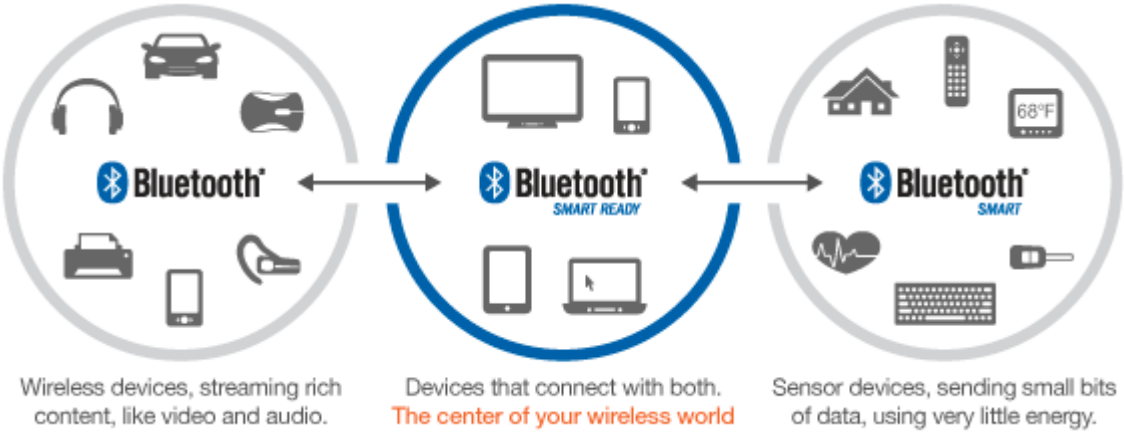
Ο βασικότερος περιορισμός του BLE είναι στο data throughput. Όπως θα δούμε στο επόμενο κεφάλαιο, η μέγιστη θεωρητική ταχύτητα μεταφοράς δεδομένων από μια συσκευή σε μία άλλη είναι αρκετά μικρή. Και, βέβαια, πρόκειται για το απόλυτο μέγιστο. Σε πραγματικά συστήματα, μπορεί να τίθενται επιπλέον περιορισμοί από το hardware ή το software, οι οποίοι να μειώνουν ακόμη περισσότερο τη μέγιστη δυνατή ταχύτητα. Με άλλα λόγια, το BLE δεν είναι κατάλληλο για εφαρμογές που απαιτούν την αποστολή μεγάλων ποσοτήτων δεδομένων σε μικρό χρονικό διάστημα. Και δεν είναι κατάλληλο ακριβώς επειδή δε σχεδιάστηκε για τέτοιες εφαρμογές, αλλά για τη μεταφορά μικρών ποσοτήτων δεδομένων, της τάξης των μερικών bytes, ανά σχετικά μεγάλα χρονικά διαστήματα (π.χ. μία φορά ανά δευτερόλεπτο). Τα δεδομένα αυτά μπορεί αν είναι δεδομένα κατάστασης, όπως το αποτέλεσμα της μέτρησης ενός αισθητήρα, ή δεδομένα εντολών, όπως μια εντολή στον αισθητήρα να πραγματοποιήσει μέτρηση άμεσα και να στείλει το αποτέλεσμα.

Ένας δεύτερος περιορισμός είναι στην εμβέλεια των συσκευών BLE. Εδώ βέβαια θα πρέπει να αναφέρουμε ότι, εξαιτίας διαφορετικού modulation και τρόπου χρήσης του φάσματος, το BLE έχει μέγιστη θεωρητική εμβέλεια μεγαλύτερη από αυτή του Bluetooth Classic (παραπάνω από 100 μέτρα σε ανοιχτό χώρο). Επειδή όμως μας ενδιαφέρει η ελαχιστοποίηση της κατανάλωσης ενέργειας, οι συσκευές σχεδιάζονται συνήθως, ώστε να μεταδίδουν στη μέγιστη ισχύ που είναι απαραίτητη για την κάθε περίπτωση χρήσης. Έτσι ένα proximity tag μπορεί να έχει εμβέλεια γύρω στα 5 έως 10 μέτρα, ενώ μία wearable συσκευή δε χρειάζεται συνήθως εμβέλεια μεγαλύτερη από 1 έως 2 μέτρα.

1.3.4 Τύποι συσκευών Bluetooth Low Energy

Στο BLE ορίζονται δύο τύποι συσκευών, με βάση την υποστήριξη που παρέχουν για επικοινωνία χρησιμοποιώντας το Bluetooth Classic ή το BLE. Οι συσκευές dual-mode μπορούν να επικοινωνούν τόσο με συσκευές Bluetooth Classic, όσο και με συσκευές BLE. Οι συσκευές single-mode μπορούν να επικοινωνούν μόνο με συσκευές BLE. Υπάρχει, βέβαια, και μία τρίτη κατηγορία, που αποτελείται από συσκευές που υποστηρίζουν μόνο το Bluetooth Classic, όπως όλες οι παλαιότερες της έκδοσης 4.0, αλλά και νέες που δε χρειάζεται να υποστηρίζουν το BLE. Οι τελευταίες δεν μπορούν να επικοινωνήσουν με συσκευές BLE.

Στην παρακάτω εικόνα βλέπουμε τα διάφορα είδη συσκευών που ανήκουν στην κάθε κατηγορία, καθώς και τις δυνατότητες επικοινωνίας μεταξύ τους.



Επίσης, όπως παρατηρούμε στην παραπάνω εικόνα, το SIG έχει δημιουργήσει δύο επιπλέον λογότυπα για τους δύο διαφορετικούς τύπους συσκευών του BLE, τα Bluetooth Smart και Bluetooth Smart Ready. Οι συσκευές, που φέρουν αυτά τα λογότυπα, πρέπει να υποστηρίζουν τις λειτουργίες που ορίζονται από το SIG για κάθε περίπτωση.

Bluetooth® SMART READY

- Dual-Mode συσκευή που υποστηρίζει το Bluetooth Core Specification Version 4.0 (ή νεότερο).
- Επιτρέπει στον χρήστη την αναβάθμιση της εφαρμογής που τρέχει στη συσκευή, για υποστήριξη επιπλέον Bluetooth Smart προϊόντων.

Bluetooth® SMART

- Single-Mode συσκευή που υποστηρίζει το Bluetooth Core Specification Version 4.0 (ή νεότερο).
- Χρησιμοποιεί την αρχιτεκτονική GATT, προκειμένου να ενεργοποιήσει τη συγκεκριμένη λειτουργικότητα της.

Η δεύτερη προϋπόθεση για τις Bluetooth Smart Ready συσκευές, υλοποιείται, για παράδειγμα, στα smartphones, με το κατέβασμα νέων εκδόσεων ή και διαφορετικών εφαρμογών, οι οποίες υποστηρίζουν νέα Bluetooth Smart προϊόντα.

1.4 Διαφορές Bluetooth Classic – Bluetooth Low Energy

Ακολουθεί ένας πίνακας που συγκρίνει επιγραμματικά τα Bluetooth Classic και BLE σε διάφορους τομείς της λειτουργίας τους.

Χαρακτηριστικό	Bluetooth Classic	Bluetooth Low Energy
RF Channels	79	40 κανάλια εύρους 2 MHz
Nodes	7 / 16777184 (parked)	Πρακτικά απεριόριστος αριθμός (> 2 δισεκατομμύρια)
Modulation	GFSK PSK (EDR)	GFSK
Modulation Index	0.25 – 0.35	0.45 – 0.55 πιο ευρύ σήμα, μεγαλύτερη ανοχή στο θόρυβο
Max Tx Power	20dBm (class 1) 4dBm (class 2)	10 dBm
Rx Sensitivity (typical)	-85 dBm	-85 dBm
Range (typical)	30 μέτρα	50 μέτρα
Packet Format	6	2
Ack Packet Len	126 μs	80 μs
8 octet Packet	214 μs	144 μs
Max Packet Size	2875 μs (1021 bytes)	328 μs (27 bytes)
Max Data Rate	2178.1 kbps	305 kbps
Time to transfer 1MB	8.81 sec (BR) 2.93 sec (EDR) < 1 sec (HS)	13.9 sec
CRC Strength	16 bit	24 bit
Encryption	Safer+	AES-128
Authentication	μία φορά	ανά πακέτο
Acknowledge	άμεση	sliding window (1 bit) lazy acknowledgement scheme

Topology	Piconet / Scatternet	Star
Discoverable	Inquiry Scanning 11.25 ms / 1.25 s	Advertising 1.25 ms / 1.25 s
Connectable	Page Scanning 11.25 ms / 1.25 s	Advertising 1.25 ms / 1.25 s
Discoverable + Connectable	Inquiry + Page Scan 22.5 ms / 1.25 s	Advertising 1.25 ms / 1.25 s
LMP PDUs	75	14
Feature bits	59	1
Connection time	20 ms	2.5ms
LMP negotiation time	minimum 5 ms ~50 ms	Δεν απαιτείται
L2CAP connection setup time	minimum 5 ms ~50 ms	Προεπιλεγμένα κανάλια Δεν απαιτείται
Time to send application data	30 ms ~ 120 ms	3 ms
Time to AFH	1.25 ms	άμεσα
Time to Sniff Subrating	2.5 ms	άμεσα
Protocols Supported by Host	14	3
Min protocols for application	3 (SDP, L2CAP, App Protocol)	2 (ATT, L2CAP)
L2CAP overhead	4 – 12 bytes	4 bytes
L2CAP configuration options	7	0
L2CAP commands	17	1

1.5 Συνοπτική περιγραφή των περιεχομένων της διπλωματικής εργασίας

Στα επόμενα δύο κεφάλαια θα ασχοληθούμε με τη στοίβα πρωτοκόλλων του Bluetooth Low Energy (BLE), όπως αυτή ορίζεται στα specifications, που είναι ελεύθερα διαθέσιμα από το Bluetooth SIG. Στο κεφάλαιο 2 θα περιγράψουμε τα κατώτερα τμήματα της στοίβας πρωτοκόλλων, καθώς και τα Generic Access Profile (GAP) και Security Manager Protocol (SMP), ενώ στο κεφάλαιο 3 θα περιγράψουμε το μοντέλο δεδομένων του BLE και το πώς αυτό χρησιμοποιείται για την υλοποίηση εφαρμογών και περιπτώσεων χρήσης.

Πιο συγκεκριμένα, στο κεφάλαιο 2, θα ασχοληθούμε με το φυσικό επίπεδο (PHY) του BLE και το επίπεδο ζεύξης δεδομένων (Link Layer – LL), τα οποία βρίσκονται στο κομμάτι του συστήματος που ονομάζεται controller. Ο controller επικοινωνεί με τα ανώτερα τμήματα, που βρίσκονται στο κομμάτι του συστήματος που ονομάζεται host, μέσω μιας διεπαφής που ονομάζεται Host Controller Interface (HCI). Το HCI ορίζεται στο specification του BLE ως ένα σύνολο από εντολές, τις οποίες μπορεί να στείλει ο host στον controller και οι οποίες χρησιμοποιούνται για την εκτέλεση διάφορων ενεργειών του πρωτοκόλλου, όπως δημιουργία σύνδεσης ή αποσύνδεση, ανταλλαγή δεδομένων ανώτερων επίπεδων, ανταλλαγή πληροφοριών σχετικά με τους peers και τις δυνατότητές τους, αρχικοποίηση κρυπτογράφησης κλπ. Επίσης ορίζονται και γεγονότα, για τα οποία ο controller μπορεί να ενημερώσει τον host μέσω του HCI. Το specification ορίζει επίσης και τα transport layers, τα οποία μπορούν να χρησιμοποιηθούν για την επικοινωνία μεταξύ host και controller. Για παράδειγμα, ορίζεται τρόπος λειτουργίας του HCI πάνω από USB ή UART.

Πάνω από το HCI, στην πλευρά του host, υπάρχει το Logical Link Control and Adaptation Protocol (L2CAP). Αυτό παρέχει στα ανώτερα επίπεδα δυνατότητες πολύπλεξης των δεδομένων τους, καθώς και τη δυνατότητα διάσπασης και ανασύνθεσης πακέτων δεδομένων, τα οποία είναι μεγαλύτερα από το μέγιστο μέγεθος δεδομένων ανά πακέτο του Link Layer. Το L2CAP υπάρχει τόσο στο Bluetooth Classic, όσο και στο BLE, αλλά η χρήση του στο δεύτερο είναι πολύ απλοποιημένη. Το Security Manager Protocol καθορίζει ρόλους και διαδικασίες με τις οποίες δύο συσκευές μπορούν να δημιουργήσουν και να ανταλλάξουν κλειδιά κρυπτογράφησης. Αυτό, με τη σειρά του, δίνει τη δυνατότητα κρυπτογράφησης των δεδομένων της σύνδεσης, πιστοποίησης αυθεντικότητας των συσκευών, καθώς και απόκρυψης της δημόσιας Bluetooth διεύθυνσης των συσκευών, για την αποφυγή της παρακολούθησής τους από κακόβουλους χρήστες. Το Generic Access Profile (GAP) αποτελεί ένα από τα βασικότερα στοιχεία του πρωτοκόλλου. Καθορίζει ρόλους και διαδικασίες με τις οποίες δύο συσκευές μπορούν να ανακαλύψουν η μία την άλλη, να συνδεθούν και να ανταλλάξουν δεδομένα, χρησιμοποιώντας το BLE. Το GAP είναι ένα από τα κοινά στοιχεία μεταξύ των Bluetooth Classic και BLE, αν και οι διαδικασίες, που χρησιμοποιούνται στο καθένα από τα δύο, είναι σε μεγάλο βαθμό διαφορετικές.

Το SMP και το GAP ανήκουν στο ανώτερο επίπεδο της στοίβας πρωτοκόλλων (στον host). Παρόλα αυτά τα περιγράφουμε μαζί με τα πρωτόκολλα του κατώτερου επίπεδου, επειδή συνδέονται σε μεγάλο βαθμό με αυτά. Για παράδειγμα, ενώ το Link Layer specification είναι αυτό που ορίζει τον τρόπο με τον οποίο μια συσκευή μπορεί να ανακοινώσει την παρουσία της σε ενδιαφερόμενες συσκευές, είναι το GAP αυτό που καθορίζει τη μορφή και το περιεχόμενο των δεδομένων που περιέχονται στα πακέτα που στέλνονται. Ένας δεύτερος λόγος είναι, προκειμένου στο επόμενο κεφάλαιο να ασχοληθούμε αποκλειστικά με το μοντέλο δεδομένων του BLE.

Στο κεφάλαιο 3, λοιπόν, ασχολούμαστε με το μοντέλο δεδομένων του BLE. Αυτό είναι ίσως το σημαντικότερο κομμάτι του πρωτοκόλλου σε ό,τι αφορά τους developers, διότι οι εφαρμογές και οι περιπτώσεις χρήσης ορίζονται πάνω σε αυτό το μοντέλο δεδομένων. Τα

κατώτερα τμήματα της στοίβας αφορούν κυρίως τους κατασκευαστές υλικού, ενώ οι ρόλοι και οι διαδικασίες που ορίζονται στα GAP και SMP, παρότι είναι καλό ένας developer να τα γνωρίζει, ώστε να ξέρει πώς λειτουργεί το όλο σύστημα, συνήθως παρουσιάζονται απλοποιημένα μέσω Application Programming Interfaces (APIs) στην εκάστοτε συσκευή. Μάλιστα, τα περισσότερα APIs συσκευών που υποστηρίζουν το BLE, είναι γραμμένα σε απευθείας συνάρτηση με το μοντέλο δεδομένων του BLE, ορίζοντας διαδικασίες οι οποίες έχουν ένα προς ένα σχέση με τις αντίστοιχες διαδικασίες του πρωτοκόλλου.

Το μοντέλο δεδομένων του BLE αποτελείται από δύο βασικά στοιχεία, το Attribute Protocol (ATT) και το Generic Attribute Profile (GATT). Το ATT περιγράφει τη μορφή των δεδομένων (attributes), που αποθηκεύονται σε μια βάση δεδομένων στη συσκευή που αποτελεί τον ATT server, καθώς και τις διαδικασίες με τις οποίες μια άλλη συσκευή, ο ATT client, μπορεί να ζητήσει αυτά τα δεδομένα. Η βάση δεδομένων έχει επίπεδη μορφή οργάνωσης, με την έννοια ότι τα δεδομένα δεν εμφανίζουν κάποια δομή, αλλά είναι απλά τοποθετημένα το ένα δίπλα στο άλλο, παρέχοντας απλώς τη δυνατότητα διευθυνσιοδότησης και ομαδοποίησης με βάση τη διεύθυνση. Σε αυτό το σημείο έρχεται το GATT, το οποίο, χτίζοντας πάνω σε αυτή τη δυνατότητα ομαδοποίησης, δημιουργεί δομές δεδομένων πάνω από την επίπεδη βάση. Οι ρόλοι στο GATT είναι ακριβώς οι ίδιοι με το ATT, δηλαδή ο GATT server είναι ATT server, και, αντίστοιχα, ο GATT client είναι ATT client, ενώ οι διαδικασίες του GATT αποτελούνται από μία ή περισσότερες διαδικασίες του ATT, σε κάποιες περιπτώσεις με σχέση ένα προς ένα. Το GATT περιγράφει πώς το ATT μπορεί να χρησιμοποιηθεί, προκειμένου να δημιουργηθεί μια ιεραρχία δεδομένων, ως μία σειρά από υπηρεσίες (services), κάθε μια από τις οποίες αποτελείται από ένα ή περισσότερα χαρακτηριστικά (characteristics). Μια υπηρεσία μπορεί να περιέχει άλλες υπηρεσίες, ορίζοντας μια ιεραρχία υπηρεσιών. Τα χαρακτηριστικά, με τη σειρά τους, αποθηκεύουν τα δεδομένα της εκάστοτε εφαρμογής, ή μπορούν να χρησιμοποιηθούν για την εκτέλεση εντολών στον server. Τα χαρακτηριστικά μπορούν επίσης να περιέχουν και περιγραφείς (descriptors), που ορίζουν μεταδεδομένα σχετικά με το χαρακτηριστικό, όπως η μονάδα μέτρησης, η μορφή (format) της τιμής (π.χ. uint8, int32, utf8s), καθώς και ρυθμίσεις σχετικές με λειτουργίες του server.

Αν εξαιρέσουμε το GAP, το οποίο, αν και συνήθως παρουσιάζεται στην κορυφή της στοίβας πρωτοκόλλων του BLE, αλληλεπιδρά με πολλούς τρόπους με όλα τα υπόλοιπα επίπεδα, το GATT βρίσκεται στο ανώτερο επίπεδο της στοίβας, και είναι αυτό με το οποίο έρχονται σε άμεση επαφή οι developers BLE εφαρμογών. Πάνω στο GATT, λοιπόν, χτίζονται τα λεγόμενα GATT-based profiles, τα οποία ορίζουν ρόλους συσκευών και υπηρεσίες που πρέπει να υποστηρίζονται από κάθε ρόλο, με σκοπό την υλοποίηση μιας περίπτωσης χρήσης. Το Bluetooth SIG παρέχει μία σύνταξη XML για τον τυπικό ορισμό των χαρακτηριστικών, των υπηρεσιών και των προφίλ (π.χ. μορφή των δεδομένων, ορισμός δυνατών τιμών, περιγραφή υπηρεσιών και ρόλων). Επίσης, παρέχει και κάποια προφίλ για διαδεδομένες περιπτώσεις χρήσης (τα επονομαζόμενα adopted profiles), στα οποία, βέβαια, εκτός από την τυπική περιγραφή μέσω XML, υπάρχει και ξεχωριστό specification για το καθένα, όπου ορίζονται με λεπτομέρεια οι λειτουργίες του προφίλ. Τα GATT-based profiles αποτελούν, κατά μία έννοια, το επίπεδο εφαρμογής του BLE, πάνω στο οποίο οι developers μπορούν να χτίσουν τις εφαρμογές τους.

Στο κεφάλαιο 4, θα ασχοληθούμε με το ενσωματωμένο σύστημα DA14580 και την αντίστοιχη αναπτυξιακή πλακέτα, πάνω στην οποία δημιουργήσαμε το κομμάτι του συστήματος, που, χρησιμοποιώντας κατάλληλους αισθητήρες, μετρά δεδομένα σχετικά με τον καιρό, τα οποία, στη συνέχεια, στέλνει μέσω BLE στην εφαρμογή που τρέχει στο smartphone. Θα περιγράψουμε τα βασικά στοιχεία από τα οποία αποτελείται το συγκεκριμένο System on a Chip (Soc), τα κυριότερα χαρακτηριστικά του firmware (ουσιαστικά ένα μικρό real-time operating system πάνω από το BLE baseband) και τις υπηρεσίες που αυτό παρέχει, καθώς και το framework που παρέχεται (σε μορφή κώδικα) μαζί με το αναπτυξιακό για τη γρηγορότερη ανάπτυξη εφαρμογών BLE.

Στο κεφάλαιο 5, θα περιγράψουμε τα βασικότερα στοιχεία προγραμματισμού εφαρμογών για το Android operating system. Το συγκεκριμένο πεδίο, βέβαια, είναι αρκετά ευρύ και, ως εκ τούτου, δεν είναι δυνατόν να περιγραφεί αναλυτικά, ούτε καν εισαγωγικά, σε ένα κεφάλαιο διπλωματικής εργασίας. Για αυτόν τον λόγο, η περιγραφή θα περιοριστεί στα βασικά στοιχεία, που αφορούν άμεσα την εφαρμογή που δημιουργήσαμε, ώστε να καταστεί ευκολότερη η κατανόηση της περιγραφής της εφαρμογής σε επόμενο κεφάλαιο.

Στο κεφάλαιο 6, περνάμε πλέον στα στοιχεία της υλοποίησης του συστήματος. Στο κεφάλαιο αυτό, θα ασχοληθούμε με την περιγραφή των χαρακτηριστικών που απαρτίζουν τις υπηρεσίες που παρέχονται από τον GATT server, στην προκειμένη περίπτωση η ενσωματωμένη εφαρμογή, και πώς αυτές μπορούν να χρησιμοποιηθούν από τον GATT client, στην προκειμένη περίπτωση το Android app, που τρέχει στο smartphone. Στην ουσία δηλαδή, θα περιγράψουμε το GATT-based profile που δημιουργήσαμε για τη συγκεκριμένη περίπτωση χρήσης, ως μία σειρά από XML αρχεία, αντίστοιχα με αυτά που ορίζονται από το SIG. Επειδή, όμως, ο τυπικός ορισμός μέσω XML δεν αρκεί για να περιγράψει όλες τις λειτουργίες και δυνατότητες του συστήματος, θα αναλύσουμε, επιπλέον, τις σχεδιαστικές επιλογές που κάναμε κατά την υλοποίηση και τον τρόπο λειτουργίας και αλληλεπίδρασης των δύο τμημάτων του συστήματος.

Στο κεφάλαιο 7, θα ασχοληθούμε με το τμήμα του συστήματος που τρέχει στην αναπτυξιακή πλακέτα. Θα περιγράψουμε τα βασικά στοιχεία της ενσωματωμένης εφαρμογής, τις σχεδιαστικές επιλογές και τον τρόπο λειτουργίας της. Θα δείξουμε πώς χρησιμοποιήσαμε το παρεχόμενο framework και τις υπηρεσίες του firmware, προκειμένου να υλοποιήσουμε το προφίλ του προηγούμενου κεφαλαίου. Τέλος, θα περιγράψουμε τον τρόπο σύνδεσης του προφίλ με τους αισθητήρες, ώστε, αυτοί, με δομημένο τρόπο, να πραγματοποιούν μετρήσεις με βάση τις παραμέτρους λειτουργίας του συστήματος, όπως καθορίζονται από τον χρήστη, και, στη συνέχεια, να επικοινωνούν με τον κώδικα υλοποίησης του προφίλ, ώστε αυτό να στέλνει τα δεδομένα στο άλλο κομμάτι του συστήματος.

Τέλος, στο κεφάλαιο 8, θα ασχοληθούμε με το κομμάτι του συστήματος που τρέχει σε smartphone ή tablet. Το κομμάτι αυτό είναι η διεπαφή του χρήστη με το σύστημα καταγραφής καιρού, οπότε μεγάλο μέρος του κεφαλαίου αφορά την περιγραφή του user interface, με το οποίο ο χρήστης μπορεί να βλέπει τα συλλεγόμενα δεδομένα, είτε αυτούσια, είτε μετά από σχετική επεξεργασία, καθώς και να καθορίζει τις παραμέτρους λειτουργίας του συστήματος. Θα περιγραφούν τα βασικά στοιχεία της εφαρμογής και οι σχεδιαστικές επιλογές που έγιναν κατά την υλοποίηση. Τέλος, θα δούμε κάποιες περιπτώσεις χρήσης της εφαρμογής με τη βοήθεια screen-shots.

1.6 Μια επισήμανση σχετικά με το Bluetooth Low Energy specification

Το specification του BLE περιέχεται, όπως αναφέραμε νωρίτερα, στην έκδοση 4.0 του Bluetooth (Core Version 4.0), η οποία έγινε αποδεκτή (adopted) ως πρότυπο από το SIG στα μέσα του 2010 και, έκτοτε, αποκτά όλο και μεγαλύτερη δημοτικότητα, καθώς όλο και περισσότερες συσκευές εμφανίζονται στην αγορά, οι οποίες ακολουθούν το συγκεκριμένο πρότυπο, είτε πρόκειται για single-mode Bluetooth Smart συσκευές, είτε πρόκειται για dual-mode Bluetooth Smart Ready συσκευές, όπως τα τελευταίας γενιάς smartphone και tablets. Κατά τα έτη 2011 έως 2013, ακολούθησαν κάποιες διορθώσεις με τρία Core Specification Addendums (CSA 2,3,4). Στα τέλη του 2013, όμως, έγινε αποδεκτή ως πρότυπο η νέα έκδοση του Bluetooth, η 4.1 (Core Version 4.1), στην οποία υπάρχουν κάποιες αλλαγές σε ό,τι αφορά το BLE. Σε πολλές περιπτώσεις, αυτές αφορούν την προτυποποίηση πρακτικών που ήταν ήδη διαδεδομένες σε πραγματικά συστήματα, την κατάργηση περιορισμών του πρωτοκόλλου ή λειτουργιών που θεωρούνταν δύσχρηστες και, ως εκ τούτου, δε χρησιμοποιούνταν συχνά στην πράξη, καθώς και την προσθήκη επιπλέον δυνατοτήτων στο πρότυπο, ώστε αυτό να μπορεί αν ανταποκριθεί στις συνεχώς αυξανόμενες απαιτήσεις των BLE developers και hardware vendors, οι οποίοι, ως μέλη του SIG, είχαν τη δυνατότητα καθορισμού του προτύπου.

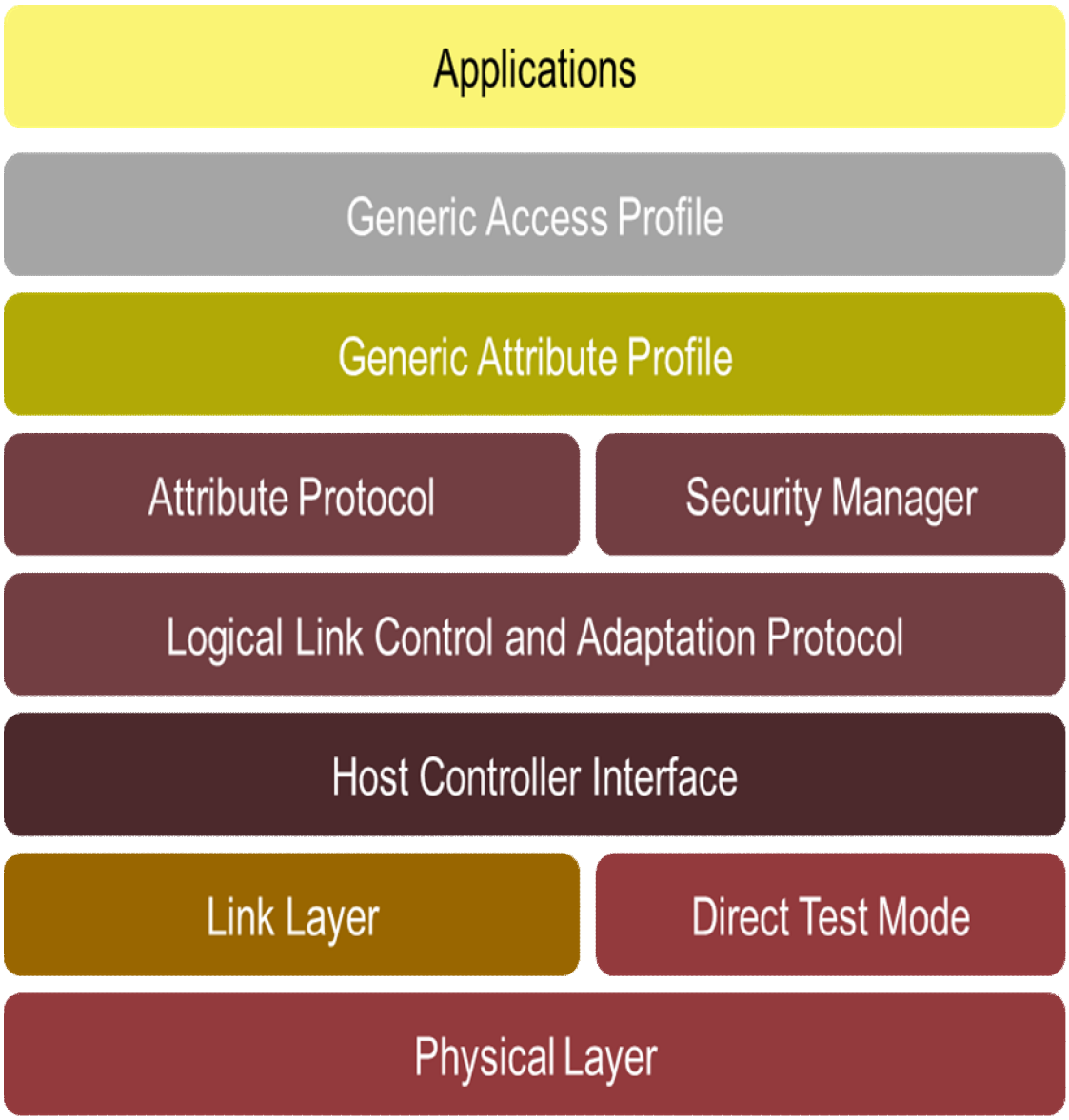
Η έκδοση 4.1 του Bluetooth είναι, βέβαια, προς τα πίσω συμβατή με την έκδοση 4.0. Παρόλα αυτά, ως νεότερη έκδοση, δεν έχει ακόμα την ίδια δημοτικότητα με την προηγούμενη. Σύμφωνα με το specification του, το DA14580 υποστηρίζει την τελευταία έκδοση του Bluetooth. Παρότι η έκδοση του Bluetooth δεν επηρεάζει με κανένα ουσιαστικό τρόπο την υλοποίηση του συστήματος, κάνουμε αυτή την επισήμανση, επειδή στα επόμενα κεφάλαια περιγράψουμε τη στοίβα πρωτοκόλλων του BLE. Η περιγραφή αυτή βασίζεται σε μεγάλο βαθμό στην έκδοση 4.0, με επιπλέον προσθήκες για σημαντικές τροποποιήσεις που έγιναν με την τελευταία έκδοση.

Κεφάλαιο 2

Bluetooth Low Energy
Controller, HCI, L2CAP, SMP, GAP

2.1 Bluetooth Low Energy Protocol Stack – Συνοπτική περιγραφή

Στην εικόνα που ακολουθεί, παρουσιάζεται η αρχιτεκτονική και τα στοιχεία από τα οποία αποτελείται η στοίβα πρωτοκόλλων του Bluetooth Low Energy. Κάνουμε εδώ μια συνοπτική αναφορά στα διάφορα στοιχεία. Περισσότερες λεπτομέρειες ακολουθούν στα αντίστοιχα κεφάλαια.



Στο κατώτερο στρώμα έχουμε το φυσικό επίπεδο (physical layer), το οποίο είναι το κύκλωμα (baseband) που αναλαμβάνει την αποστολή και λήψη πακέτων. Πάνω από αυτό υπάρχει το επίπεδο ζεύξης δεδομένων (link layer) και το Direct Test Mode. Το τελευταίο αποτελεί ένα πρότυπο τρόπο, με τον οποίο μπορεί να ελεγχθεί η σωστή λειτουργία του φυσικού επίπεδου. Άλλες τεχνολογίες ασύρματης μεταφοράς δεν έχουν αντίστοιχη δυνατότητα δοκιμών στο φυσικό επίπεδο. Τα στοιχεία αυτά αποτελούν το κατώτερο κομμάτι του συστήματος, το οποίο ονομάζεται Controller.

Τα υπόλοιπα στοιχεία του συστήματος, που βρίσκονται στα ανώτερα επίπεδα, αποτελούν το κομμάτι που ονομάζεται Host. Μεταξύ του host και του controller υπάρχει μία διεπαφή, η οποία έχει το (προφανές) όνομα Host Controller Interface (HCI). Αυτό αποτελείται από μια σειρά προτυποποιημένων εντολών, γεγονότων και διαδικασιών, μέσω των οποίων επιτυγχάνεται η επικοινωνία μεταξύ των host και controller, προκειμένου να εκτελεστούν οι διάφορες λειτουργίες του πρωτοκόλλου. Στο Bluetooth specification δεν προτυποποιείται απλώς το HCI, αλλά ορίζονται επίσης και συγκεκριμένα transport layers, τα οποία μπορούν να χρησιμοποιηθούν για τη σύνδεση των host και controller, και τη μεταφορά των εντολών και δεδομένων του HCI. Παραδείγματα transport layers είναι τα USB και UART. Η προτυποποίηση του HCI μπορεί να οδηγήσει σε αρχιτεκτονικές, όπου ο host και ο controller βρίσκονται σε διαφορετικά chipset (split architecture), τα οποία μπορεί να παράγονται ακόμα και από διαφορετικούς κατασκευαστές, και τα οποία μπορούν να επικοινωνούν, μέσω του πρότυπου τρόπου που ορίζεται στο specification. Αυτού του είδους οι αρχιτεκτονικές χρησιμοποιούνται κυρίως από Bluetooth Smart Ready συσκευές, στις οποίες υπάρχει μια CPU με μεγάλη υπολογιστική ισχύ, όπου έχει νόημα η υλοποίηση του host ως software που θα τρέχει στον κεντρικό επεξεργαστή. Στις Bluetooth Smart συσκευές, λόγω παραγόντων κόστους και μεγέθους, προτιμάται συνήθως η λογική του System on a Chip (SoC), όπου το HCI υλοποιείται εσωτερικά στο σύστημα. Σε ορισμένες περιπτώσεις, χρησιμοποιείται ένας επιπλέον διαχωρισμός, στον οποίο το application layer τρέχει επίσης σε ξεχωριστό επεξεργαστή. Έτσι, μπορούμε να έχουμε αρχιτεκτονικές ακόμα και τριών διαφορετικών chipset. Αυτός ο τρόπος, όμως, δεν περιέχεται στο πρότυπο, οπότε χρησιμοποιεί ιδιωτικά πρωτόκολλα για την επικοινωνία των layers. Με τις διαφορετικές δυνατότητες οργάνωσης του συστήματος, θα ασχοληθούμε ξανά σε επόμενο κεφάλαιο, όπου θα δούμε πώς αυτοί οι τρόποι υλοποιούνται από το DA14580.

Πάνω από το HCI, στην πλευρά του host, υπάρχει το Logical Link Control and Adaptation Protocol (L2CAP). Το L2CAP παρέχει υπηρεσίες πολύπλεξης στα πρωτόκολλα των ανώτερων στρωμάτων, καθώς και δυνατότητα διάσπασης και ανασύνθεσης πακέτων δεδομένων που δεν μπορούν να σταλούν σε ένα πακέτο του Link Layer. Πάνω από αυτό έχουμε τον Security Manager (SM) και το αντίστοιχο πρωτόκολλο (SMP). Το Security Manager Protocol ορίζει τις διαδικασίες, με τις οποίες μπορεί γίνει παραγωγή και ανταλλαγή κλειδιών μεταξύ των peers μιας σύνδεσης, προκειμένου να εκτελεστούν λειτουργίες όπως κρυπτογράφηση, πιστοποίηση αυθεντικότητας, διασφάλιση απορρήτου, και, βέβαια, σημαντικές λειτουργίες του Bluetooth, όπως pairing και bonding.

Το Attribute Protocol (ATT) και το Generic Attribute Profile (GATT) ορίζουν το μοντέλο δεδομένων του BLE. Είναι τα κομμάτια εκείνα (ειδικά το GATT), με τα οποία ένας developer έρχεται άμεσα σε επαφή. Τα GATT-based profiles, που ορίζονται πάνω από το GATT, αποτελούν ουσιαστικά τον τυπικό ορισμό του application layer για μια συγκεκριμένη περίπτωση χρήσης. Τα ATT και GATT περιγράφονται με λεπτομέρεια στο επόμενο κεφάλαιο.

Τέλος, έχουμε το Generic Access Profile (GAP). Το GAP ορίζει ρόλους, τρόπους λειτουργίας (modes) και διαδικασίες (procedures), με τις οποίες δύο συσκευές μπορούν να ανακαλύψουν η μία την άλλη, να συνδεθούν και να ανταλλάξουν δεδομένα χρησιμοποιώντας το BLE. Το GAP είναι ένα από τα σημαντικότερα στοιχεία της στοίβας πρωτοκόλλων του Bluetooth, το οποίο διασυνδέεται με ποικίλους τρόπους με τα υπόλοιπα.

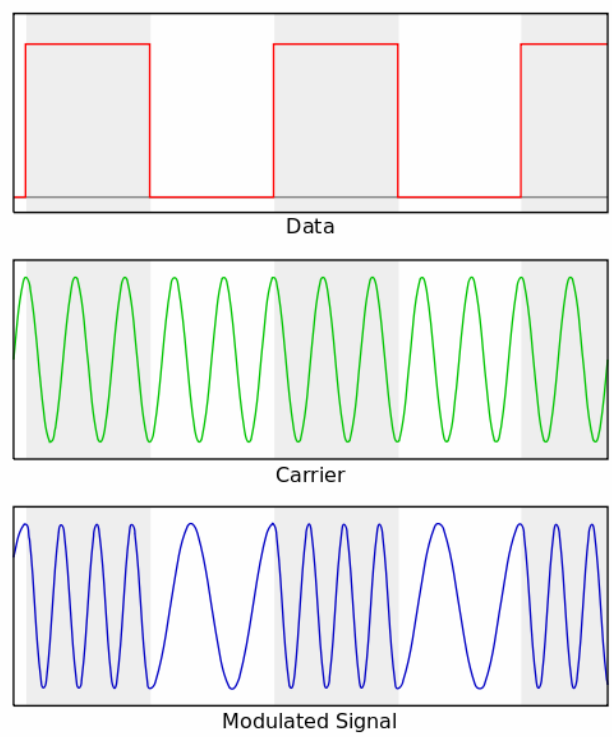
Όπως αναφέραμε και στην εισαγωγή, σε αυτό το κεφάλαιο περιγράφουμε τον controller, το HCI και τα L2CAP, SMP και GAP, που είναι τα κομμάτια του συστήματος που δεν αφορούν άμεσα τους developers BLE εφαρμογών. Αφήνουμε, έτσι, την περιγραφή του μοντέλου δεδομένων του BLE, με το οποίο έρχονται σε άμεση επαφή οι developers, για το επόμενο κεφάλαιο.

2.2 Το φυσικό επίπεδο

Το Bluetooth Low Energy χρησιμοποιεί, όπως και το Bluetooth Classic, το επονομαζόμενο ISM (industrial, scientific and medical) εύρος συχνοτήτων, το οποίο εκτείνεται από τα 2.4 GHz μέχρι τα 2.4835 GHz (τμήμα του Ultra high frequency – UHF band). Το συγκεκριμένο εύρος συχνοτήτων είναι ελεύθερο προς χρήση διεθνώς, οπότε δε χρειάζονται ειδικές άδειες για τη χρήση του. Σε αντίθεση, όμως, με το Bluetooth Classic, το BLE χρησιμοποιεί μικρότερο αριθμό καναλιών, 40 αντί 79. Το εύρος κάθε καναλιού είναι 2 MHz, ενώ προβλέπεται guard band και στα δύο άκρα του φάσματος, 2 MHz στο μικρότερο και 3.5 MHz στο μεγαλύτερο. Αυτό χρησιμεύει στο να μειώνει τις παρεμβολές στο Bluetooth από τεχνολογίες που χρησιμοποιούν τα γειτονικά του ISM φάσματα, αλλά και τις παρεμβολές του ίδιου του Bluetooth στα γειτονικά φάσματα.

$$f_c = 2402MHz + 2k, k = 0,1,...,39$$

Η αύξηση του εύρους των καναλιών δίνει στο BLE μεγαλύτερη ανοχή στο θόρυβο, κάτι που βοηθάει στη μείωση της απαιτούμενης ισχύος εκπομπής. Το modulation που χρησιμοποιείται είναι το Frequency Shift Keying (FSK), το οποίο είναι μια ψηφιακή κωδικοποίηση, στην οποία δυαδικά δεδομένα κωδικοποιούνται με βάση την απόκλιση από μία κεντρική συχνότητα (carrier), όπως φαίνεται στο παρακάτω σχήμα.



Στην περίπτωση του BLE, η συχνότητα του carrier είναι η κεντρική συχνότητα του εκάστοτε καναλιού, ενώ η απόκλιση από αυτήν είναι τουλάχιστον 185 kHz.

$$bit = 1 \Rightarrow f = f_c + \Delta$$
$$bit = 0 \Rightarrow f = f_c - \Delta$$
$$\Delta \geq 185kHz$$

Το BLE χρησιμοποιεί μία παραλλαγή του FSK, το Gaussian FSK (GFSK). Το GFSK χρησιμοποιεί ένα Gaussian φίλτρο για την ομαλοποίηση των μεταβάσεων μεταξύ των διαφορετικών συχνοτήτων κωδικοποίησης των bit. Το φίλτρο αυτό αποτρέπει τις απότομες μεταβάσεις, χωρίς να αυξάνει υπερβολικά το χρόνο μετάβασης. Έτσι αποτρέπονται περιπτώσεις inter-symbol interference, που θα μπορούσαν να προκύψουν στον δέκτη από τις απότομες μεταβάσεις του σήματος.

Εξαιτίας του τρόπου κωδικοποίησης των bit, δεν είναι εφικτό να σταλούν μεγάλες σειρές από μηδενικά ή άσσους. Αυτό συμβαίνει επειδή, αν στέλνονταν τέτοιες σειρές, ο δέκτης δε θα μπορούσε πλέον να καταλάβει αν ο πομπός πράγματι μεταδίδει δεδομένα, ή απλώς έχει μετατοπίσει την κεντρική του συχνότητα. Το τελευταίο συμβαίνει στην πράξη, και προβλέπεται από το πρότυπο μια ανοχή μέχρι 150kHz απόκλισης από την κεντρική συχνότητα. Για να αποφευχθεί η αποστολή μεγάλων σειρών από όμοια bits, χρησιμοποιείται η τεχνική του data whitening. Σύμφωνα με αυτήν, τα δεδομένα περνούν μέσα από μια γεννήτρια ψευδοτυχαίων αριθμών, η οποία παράγει μηδενικά και άσσους με γνωστή σειρά. Έτσι, ο δέκτης των δεδομένων μπορεί να αναπαράγει την αρχική ακολουθία bits. Για τη διαδικασία αυτή χρησιμοποιείται ένας linear feedback shift register με πολυώνυμο:

$$x^7 + x^4 + x^0$$

Το data whitening γίνεται στο link layer, αμέσως μετά τον υπολογισμό του CRC. Το αναφέρουμε όμως εδώ, επειδή ο λόγος, που η χρήση του καθίσταται απαραίτητη, σχετίζεται άμεσα με το φυσικό επίπεδο.

Ένα σημαντικό μέγεθος της κωδικοποίησης με βάση τη συχνότητα είναι το modulation index, το οποίο ορίζεται ως εξής (όπου Δf η μέγιστη απόκλιση από την κεντρική συχνότητα και f_m η μέγιστη συχνότητα του αρχικού σήματος):

$$h = \frac{\Delta f}{f_m}$$

Το modulation index αποτελεί ένα μέτρο του πόσο το κωδικοποιημένο σήμα μεταβάλλεται σε σχέση με το αρχικό. Στην περίπτωση που το h είναι ίσο με 0.5, έχουμε το Minimum Shift Keying (MSK), μια ειδική περίπτωση του FSK, η οποία είναι πολύ αποδοτική ως προς τη χρήση του φάσματος. Η επίτευξη του απόλυτου MSK θα απαιτούσε ακριβότερα κυκλώματα, οπότε για το BLE επελέγη ένα modulation index μεταξύ 0.45 και 0.55, το οποίο, παρότι δεν είναι MSK, διατηρεί σε μεγάλο βαθμό τα καλά χαρακτηριστικά του.

Η ισχύς εκπομπής έχει μέγιστη τιμή τα 10 dBm (10 mW), ενώ προβλέπεται και ελάχιστη τιμή στα -20 dBm (10 μW), ώστε μία συσκευή να μη μεταδίδει τόσο χαμηλό σήμα, που να μην είναι ορατή σε γειτονικές συσκευές. Στην πλευρά του δέκτη ορίζεται, όπως και στο Bluetooth Classic, επίπεδο ευαισθησίας λήψης μικρότερο ή ίσο των -70dBm για bit error rate (BER) ίσο με 0.1%. Το παραπάνω σημαίνει ότι ένας δέκτης θα πρέπει να μπορεί να λαμβάνει σήμα -70dBm (100pW) και να το αποκωδικοποιεί με μέγιστη ανοχή το ένα λάθος ανά χίλια bit. Στην πράξη, πολλοί controllers μπορούν να αποκωδικοποιούν, με χαμηλό BER, σήμα που φτάνει μέχρι και τα -90dBm (1pW).

Η ταχύτητα εκπομπής είναι 1 Mbps, που σημαίνει ότι κάθε bit «διαρκεί» 1 μs. Η επιλογή της συγκεκριμένης ταχύτητας αποτελεί ακόμα ένα σημείο, στο οποίο ο στόχος της χαμηλής κατανάλωσης ενέργειας υπήρξε σημαντικός παράγοντας. Όταν το φυσικό επίπεδο στέλνει ή λαμβάνει δεδομένα, η ενέργεια που απαιτείται είναι σχετικά ανεξάρτητη της ταχύτητας μετάδοσης. Αυτό σημαίνει ότι η ταχύτητα μετάδοσης θα πρέπει να είναι αρκετά μεγάλη, ώστε να μειώνεται ο χρόνος αποστολής και λήψης. Από την άλλη, αν η ταχύτητα

γίνει πολύ μεγάλη, με χρήση, για παράδειγμα, πολύπλοκης κωδικοποίησης, θα απαιτείται περισσότερη ενέργεια για την κωδικοποίηση και αποκωδικοποίηση των bits. Έτσι, έχουμε ένα συμβιβασμό μεταξύ των δύο, με την απλή κωδικοποίηση και την επιλεγμένη ταχύτητα του 1 Mbps.

2.3 Το επίπεδο ζεύξης δεδομένων (Link Layer)

Το επίπεδο ζεύξης δεδομένων στο BLE, όπως και σε άλλες στοίβες πρωτοκόλλων δικτύου, ορίζει τον τρόπο με τον οποίο δύο συσκευές μπορούν να χρησιμοποιήσουν το φυσικό επίπεδο, προκειμένου να επικοινωνήσουν μεταξύ τους.

2.3.1 Η μηχανή καταστάσεων

Η λειτουργία του επιπέδου ζεύξης δεδομένων του BLE μπορεί να περιγραφεί με τη βοήθεια μίας μηχανής καταστάσεων, η οποία αποτελείται από πέντε καταστάσεις:

1. **Standby State**: Σε αυτή την κατάσταση, το Link Layer δεν εκπέμπει ούτε λαμβάνει δεδομένα. Το Link Layer μπορεί να περάσει σε αυτή την κατάσταση από οποιαδήποτε από τις άλλες καταστάσεις. Είναι η κεντρική κατάσταση της μηχανής καταστάσεων.

2. **Advertising State**: Σε αυτή την κατάσταση, το Link Layer εκπέμπει δεδομένα advertising, χρησιμοποιώντας τα advertising κανάλια, τα οποία περιγράφονται παρακάτω. Επίσης, μπορεί να ελέγχει το μέσο για απαντήσεις στα advertising πακέτα που μόλις έστειλε, οι οποίες στέλνονται και αυτές στα advertising κανάλια, και, συγκεκριμένα, στο ίδιο κανάλι όπου έγινε η αρχική μετάδοση. Μια συσκευή με το Link Layer σε Advertising state ονομάζεται advertiser. Η κατάσταση αυτή μπορεί να ακολουθήσει μόνο την κατάσταση Standby. Μετά την κατάσταση αυτή, το Link Layer μπορεί, είτε να επιστρέψει στην κατάσταση Standby, είτε να περάσει στην κατάσταση Connection (ως slave).

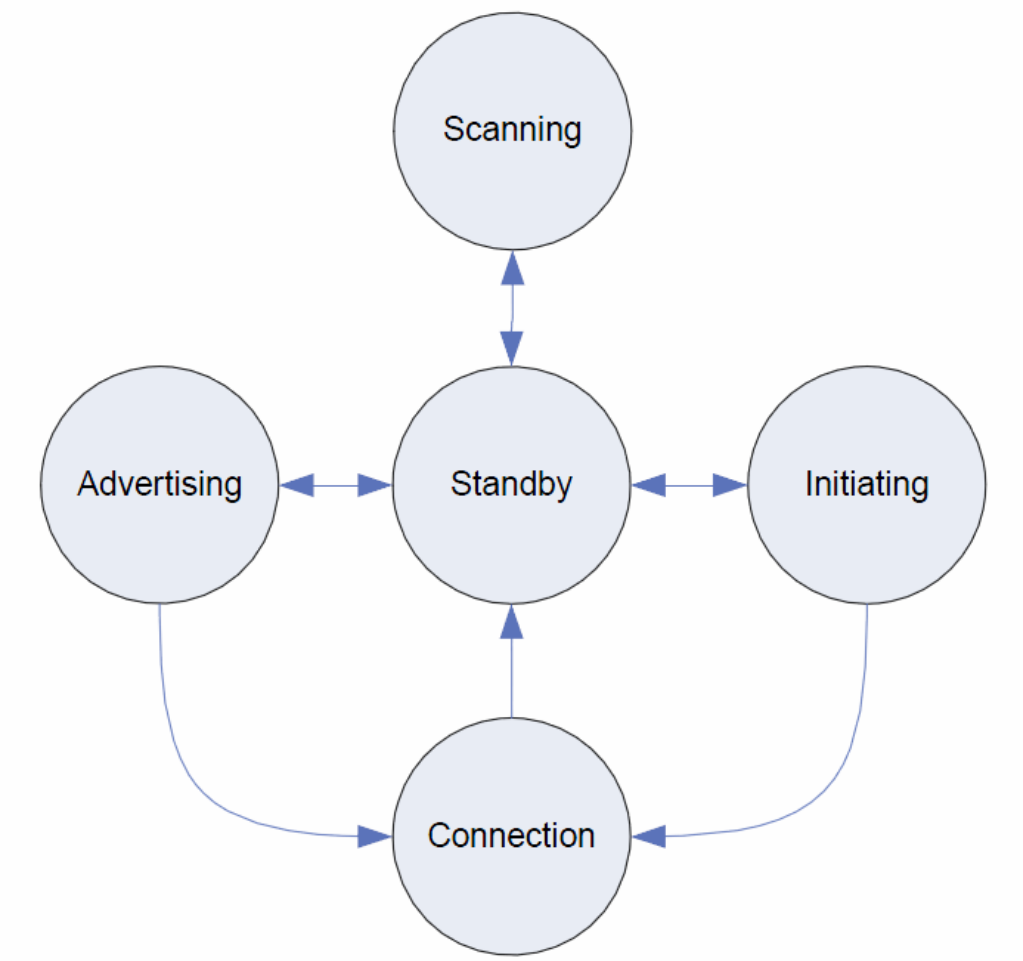
3. **Scanning State**: Σε αυτή την κατάσταση, το Link Layer ελέγχει τα advertising κανάλια, για τυχόν advertising πακέτα από συσκευές που βρίσκονται σε Advertising State. Μια συσκευή με το Link Layer σε Scanning state ονομάζεται scanner. Η κατάσταση αυτή μπορεί να ακολουθήσει μόνο την κατάσταση Standby.

4. **Initiating State**: Σε αυτή την κατάσταση, το Link Layer ελέγχει, επίσης, τα advertising κανάλια, για τυχόν advertising πακέτα. Σε αυτή την περίπτωση, όμως, ελέγχει για πακέτα από συγκεκριμένη συσκευή, με σκοπό να απαντήσει, στέλνοντας κατάλληλα πακέτα, προκειμένου να εκκινήσει μια σύνδεση με τη συσκευή αυτή. Μια συσκευή με το Link Layer σε Initiating state ονομάζεται initiator. Η κατάσταση αυτή μπορεί να ακολουθήσει μόνο την κατάσταση Standby. Μετά την κατάσταση αυτή, το Link Layer μπορεί, είτε να επιστρέψει στην κατάσταση Standby, είτε να περάσει στην κατάσταση Connection (ως master).

5. **Connection State**: Η κατάσταση αυτή μπορεί να ακολουθήσει τις καταστάσεις Initiating ή Advertising. Στην κατάσταση αυτή, η συσκευή θεωρείται ότι βρίσκεται σε σύνδεση. Μέσα σε αυτή την κατάσταση ορίζονται οι ρόλοι του master και του slave. Όπως είδαμε παραπάνω, master είναι η συσκευή που περνάει σε Connection state από το Initiating state, ενώ slave είναι η συσκευή που περνάει σε Connection state από το Advertising state. Το Link Layer σε ρόλο master επικοινωνεί με μία

συσκευή, η οποία βρίσκεται σε ρόλο slave, καθορίζοντας τις παραμέτρους της σύνδεσης. Το Link Layer σε ρόλο slave μπορεί να επικοινωνεί με μία μόνο συσκευή με το ρόλο του master.

Στην ακόλουθη εικόνα φαίνεται η μηχανή καταστάσεων του Link Layer, με τις πέντε καταστάσεις και τις δυνατές μεταβάσεις μεταξύ τους.



Η μηχανή καταστάσεων του Link Layer μπορεί να βρίσκεται σε μία μόνο κατάσταση. Επιτρέπεται, όμως, να υπάρχουν περισσότερες της μίας μηχανές καταστάσεων, κάθε μία από τις οποίες μπορεί να βρίσκεται σε διαφορετική κατάσταση. Στην αρχική έκδοση του BLE (4.0), υπήρχαν κάποιοι περιορισμοί ως προς τις καταστάσεις, στις οποίες οι διάφορες μηχανές καταστάσεων μπορούσαν να βρίσκονται ταυτόχρονα, όπως φαίνεται στον πίνακα που ακολουθεί. Με λίγα λόγια, μια συσκευή δεν επιτρεπόταν να είναι ταυτόχρονα master και slave, ενώ μια συσκευή ήδη σε ρόλο slave δεν μπορούσε να είναι ταυτόχρονα slave σε άλλη σύνδεση. Δηλαδή, ένας master είχε τη δυνατότητα να συνδέεται με πολλούς slaves, αλλά ένας slave δεν μπορούσε να συνδέεται παρά μόνο με έναν master. Οι περιορισμοί στις άλλες καταστάσεις ορίζονται με βάση τα παραπάνω. Για παράδειγμα ο master μπορεί να κάνει advertising, αλλά όχι τέτοιο, που θα μπορούσε να εκκινήσει μία σύνδεση, αφού στη σύνδεση αυτή, η συγκεκριμένη συσκευή θα γινόταν slave. Το ίδιο ισχύει για όλες τις περιπτώσεις που σημειώνονται με αστερίσκο. Το scanning επιτρέπεται σε όλες τις περιπτώσεις, εκτός, όπως και για το advertising, αν η συσκευή είναι ήδη στην κατάσταση αυτή.

Multiple State Machine State and Roles		Advertising	Scanning	Initiating	Connection	
					Master Role	Slave Role
Advertising		Prohibited	Allowed	Allowed*	Allowed*	Allowed*
Scanning		Allowed	Prohibited	Allowed	Allowed	Allowed
Initiating		Allowed*	Allowed	Prohibited	Allowed	Prohibited
Connection	Master Role	Allowed*	Allowed	Allowed	Allowed	Prohibited
	Slave Role	Allowed*	Allowed	Prohibited	Prohibited	Prohibited

Στην έκδοση 4.1 του Bluetooth, οι παραπάνω περιορισμοί καταργήθηκαν. Πλέον μια συσκευή μπορεί να βρίσκεται ταυτόχρονα σε οποιαδήποτε από τις καταστάσεις. Έτσι ένας master σε μία σύνδεση, μπορεί να είναι slave σε άλλη σύνδεση, ενώ ένας slave μπορεί να έχει περισσότερες της μίας συνδέσεις.

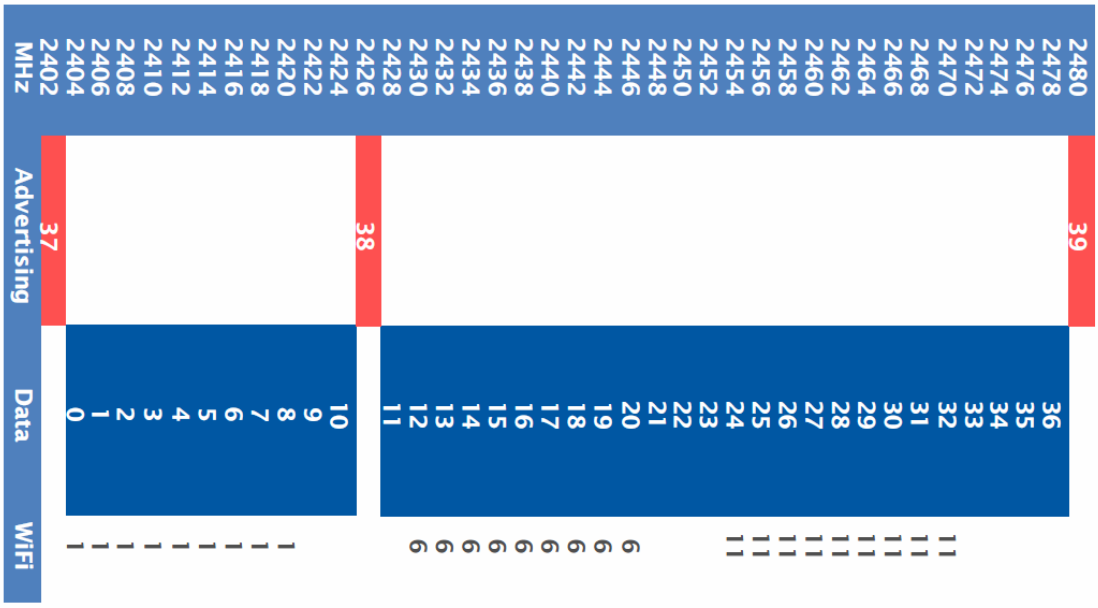
2.3.2 Διευθύνσεις – White List

Οι συσκευές Bluetooth αναγνωρίζονται από τη διεύθυνσή τους, η οποία είναι ένας αριθμός 48 bit, παρόμοιος με αυτόν που χρησιμοποιείται για τις Ethernet MAC διευθύνσεις. Υπάρχουν δύο είδη διευθύνσεων. Οι public διευθύνσεις και οι random διευθύνσεις. Μια συσκευή πρέπει να έχει τουλάχιστον μία από τις δύο, ενώ μπορεί να περιέχει και τις δύο. Οι public διευθύνσεις προγραμματίζονται κατά την παραγωγή της συσκευής και χρησιμοποιούν συγκεκριμένες ακολουθίες, τις οποίες η κατασκευάστρια εταιρία έχει αποκτήσει από το IEEE Registration Authority. Οι random διευθύνσεις μπορούν επίσης να προγραμματιστούν κατά την παραγωγή, ή να δημιουργούνται δυναμικά, όταν η συσκευή τίθεται σε λειτουργία.

Χρησιμοποιώντας τις διευθύνσεις των απομακρυσμένων συσκευών, μια συσκευή BLE μπορεί να διατηρεί στο Link Layer μία white list με συσκευές. Η λίστα αυτή χρησιμοποιείται σε διάφορες διαδικασίες του πρωτοκόλλου. Για παράδειγμα, ένας advertiser μπορεί να ρυθμιστεί, ώστε να απαντά σε scan ή connection requests, μόνο από συσκευές που περιέχονται στη λίστα. Επίσης, ένας scanner μπορεί να ρυθμιστεί, ώστε να ελέγχει τα advertising πακέτα, μόνο από συσκευές της λίστας, και, αντίστοιχα, ο initiator, να συνδέεται μόνο με συσκευές που περιέχονται σε αυτή. Στην τελευταία περίπτωση δίνεται και η δυνατότητα αυτόματης σύνδεσης.

2.3.3 Κανάλια

Όπως είδαμε στην περιγραφή του φυσικού επιπέδου, το BLE χωρίζει το ISM φάσμα σε 40 κανάλια εύρους 2MHz. Τα κανάλια αυτά διαχωρίζονται στο Link Layer σε δύο τύπους, τα advertising κανάλια, 3 τον αριθμό (με αριθμούς 37, 38 και 39), και τα data κανάλια, τα υπόλοιπα 37 (με αριθμούς από 0 έως 36). Οι θέσεις των καναλιών στο φάσμα είναι προκαθορισμένες και επιλεγμένες με τέτοιο τρόπο, ώστε το πρωτόκολλο να μην κινδυνεύει να καταστεί ανενεργό από παρεμβολές άλλων συσκευών που χρησιμοποιούν το ISM φάσμα, όπως για παράδειγμα το WiFi, του οποίου τα κυριότερα κανάλια (1, 6 και 11) καλύπτουν συγκεκριμένες περιοχές του φάσματος.



Όπως βλέπουμε στην παραπάνω εικόνα, τα τρία advertising κανάλια είναι διάσπαρτα μέσα στο ISM φάσμα και τοποθετημένα όσο το δυνατόν πιο μακριά από τα συνηθισμένα κανάλια του WiFi, της πιο διαδομένης τεχνολογίας που χρησιμοποιεί επίσης το ISM. Τα advertising κανάλια είναι πολύ σημαντικά για τη λειτουργία του BLE. Για την ακρίβεια, χωρίς αυτά, καμία συσκευή δε θα μπορούσε να ανακαλύψει άλλες συσκευές, οπότε δε θα ήταν δυνατή η δημιουργία συνδέσεων. Η συγκεκριμένη επιλογή έγινε με τέτοιο τρόπο, ώστε να εξασφαλιστεί ότι τουλάχιστον ένα από τα advertising κανάλια θα είναι διαθέσιμο. Στις συνηθισμένες περιπτώσεις είναι διαθέσιμα και τα τρία, ακόμα και αν χρησιμοποιούνται όλα τα βασικά κανάλια του WiFi. Η επιλογή του αριθμού των καναλιών είναι επίσης ένας συμβιβασμός, μεταξύ της στιβαρότητας του πρωτοκόλλου και της χαμηλής κατανάλωσης ενέργειας. Τα advertising κανάλια, όπως θα δούμε παρακάτω χρησιμοποιούνται συνήθως από Bluetooth Smart συσκευές (περιφερειακά), με σκοπό αυτές να δηλώσουν την παρουσία τους σε ενδιαφερόμενους peers. Αν, λοιπόν, είχαμε περισσότερα advertising κανάλια, ναι μεν το πρωτόκολλο θα ήταν πιο στιβαρό, αλλά οι συσκευές αυτές, οι οποίες λειτουργούν συνήθως με μπαταρία, θα έπρεπε να σπαταλούν περισσότερη ενέργεια προκειμένου να μεταδίδουν στα επιπλέον κανάλια.

Τα data κανάλια χρησιμοποιούνται για τη μεταφορά δεδομένων, από συσκευές που βρίσκονται σε σύνδεση. Προκειμένου να παρακάμψει τις παρεμβολές από όλες τις υπόλοιπες συσκευές, που χρησιμοποιούν την ίδια περιοχή του φάσματος, το BLE χρησιμοποιεί την τεχνική του Adaptive Frequency Hopping. Το Frequency Hopping είναι μία τεχνική σύμφωνα με την οποία η συχνότητα μετάδοσης, άρα και το κανάλι, αλλάζει ανά τακτά χρονικά διαστήματα, προκειμένου να αποφεύγονται απότομες (burst) παρεμβολές, όπως είναι συνήθως η κίνηση του WiFi. Για παράδειγμα, έστω ότι το BLE χρησιμοποιεί το κανάλι 15 με παρόν κάποιο δίκτυο WiFi στο κανάλι 6. Αν καμία συσκευή δε μεταδίδει δεδομένα εκείνη τη στιγμή στο δίκτυο WiFi, το BLE μπορεί να χρησιμοποιεί ελεύθερα το συγκεκριμένο κανάλι. Με το που θα ξεκινήσει η μετάδοση, όμως, όλη η σειρά των καναλιών από 12 έως και 20 καθίσταται μη διαθέσιμη. Με το Frequency Hopping, το BLE μπορεί να προλάβει τέτοιες καταστάσεις προσπερνώντας γρήγορα τα μη διαθέσιμα κανάλια. Μάλιστα δε χρειάζεται να περιμένει πρώτα να συμβεί το πρόβλημα, αφού η αλλαγή συχνότητας συμβαίνει ανεξάρτητα από τις παρεμβολές. Το Adaptive Frequency Hopping, με τη σειρά του, λαμβάνει υπόψη και την πραγματική κατάσταση του φάσματος από πλευράς παρεμβολών, με σκοπό να αποφεύγονται περιοχές του φάσματος, που εκείνη τη στιγμή είναι μη διαθέσιμες. Αυτό γίνεται με remapping των μη διαθέσιμων καναλιών στα διαθέσιμα, με κατάλληλο τρόπο,

ώστε σε κάθε αλλαγή συχνότητας να έχουμε μετάβαση σε διαφορετικό μέρος του φάσματος. Η εξίσωση που δίνει την επόμενη συχνότητα είναι η ακόλουθη:

$$f_{n+1} = (f_n + hop) \bmod 37$$

Το 37 είναι πρώτος αριθμός και, ως εκ τούτου, η συγκεκριμένη ακολουθία περνάει από όλα τα data κανάλια κάθε 37 βήματα. Το hop είναι η παράμετρος εκείνη που καθορίζει πόσο μακριά θα είναι το επόμενο κανάλι από το τρέχον. Η τιμή αυτή κυμαίνεται από 5 έως 16 και αποτελεί παράμετρο της σύνδεσης.

2.3.4 Δομή του πακέτου

Η δομή του πακέτου του BLE παρουσιάζεται στο ακόλουθο σχήμα. Τα μεγέθη είναι σε bits.

8	32	16	advertising: 0 – 296, data: 0 – 248	24
Preamble	Access Address	Header	Data	CRC

Υπάρχουν δύο είδη πακέτων στο BLE, τα advertising πακέτα και τα πακέτα δεδομένων. Ο διαχωρισμός μεταξύ τους γίνεται, όχι με βάση κάποια σημαία στο ίδιο το πακέτο, αλλά, με βάση το κανάλι στο οποίο μεταδίδονται. Τα advertising πακέτα μπορούν να μεταδίδονται μόνο στα advertising κανάλια και, αντίστοιχα, τα πακέτα δεδομένων μπορούν να μεταδίδονται μόνο στα data κανάλια.

Τα πακέτα, όπως είδαμε στην περιγραφή του φυσικού επιπέδου, μεταδίδονται bit προς bit. Σε κάθε byte δεδομένων του πακέτου, είναι το λιγότερο σημαντικό bit (Least Significant Bit – LSB) αυτό που μεταδίδεται πρώτο. Στις περιπτώσεις που έχουμε δεδομένα που αποτελούνται από περισσότερα του ενός byte, αυτά μεταδίδονται με το λιγότερο σημαντικό byte πρώτο. Έτσι η ακολουθία 0x123456 θα μεταδοθεί ως:

0110 1010 0010 1100 0100 1000

Παρατηρούμε ότι το μέγεθος των δεδομένων που περιέχονται στο πακέτο του Link Layer είναι σχετικά μικρό. Αυτό οφείλεται εν μέρει και στο γεγονός ότι το BLE σχεδιάστηκε για τη μεταφορά μικρών ποσοτήτων δεδομένων, όμως, η επιλογή αυτή επηρεάστηκε από παράγοντες κόστους, που έχουν να κάνουν με τη διαδικασία κατασκευής των κυκλωμάτων μετάδοσης. Τα κυκλώματα αυτά παράγονται συνήθως με χρήση της τεχνολογίας CMOS. Κατά τη χρήση τους, υπάρχει περίπτωση να αυξηθεί η θερμοκρασία τους, με αποτέλεσμα να μεταβληθεί η συχνότητα λειτουργίας τους. Προκειμένου να διατηρηθεί η συχνότητα σταθερή, απαιτούνται επιπλέον κυκλώματα, τα οποία αυξάνουν το κόστος κατασκευής και την κατανάλωση ενέργειας. Με το μικρό μήκος πακέτου του BLE, εξασφαλίζεται ότι τα κυκλώματα αποστολής και λήψης δε θα δουλεύουν για μεγάλα χρονικά διαστήματα, οπότε η μεταβολή στη θερμοκρασία και τη συχνότητά τους διατηρείται σε ανεκτά επίπεδα.

2.3.4.1 Κοινά στοιχεία

Το preamble είναι μια ακολουθία από 8 bits που τοποθετείται στην αρχή κάθε πακέτου. Η ακολουθία αυτή είναι της μορφής 01010101 ή 10101010, δηλαδή μια ακολουθία εναλλασσόμενων bits. Το ποια από τις δύο θα χρησιμοποιηθεί εξαρτάται από το πρώτο bit του access address, το οποίο ακολουθεί. Επιλέγεται αυτή, στην οποία το τελευταίο bit είναι διαφορετικό από το πρώτο του access address. Η χρησιμότητα του preamble είναι ότι επιτρέπει στον δέκτη να αναγνωρίσει τις συχνότητες που χρησιμοποιούνται από τον πομπό για την αναπαράσταση των bit (λόγω της επιτρεπόμενης απόκλισης που είδαμε παραπάνω), καθώς και να ανιχνεύσει την ισχύ του λαμβανόμενου σήματος, ρυθμίζοντας κατάλληλα το κέρδος (gain) λήψης.

Αμέσως μετά ακολουθεί το access address με μέγεθος 32 bits. Το access address χρησιμοποιείται για τη διάκριση μεταξύ διαφορετικών συνδέσεων στα data κανάλια. Στα advertising πακέτα υπάρχει επίσης access address, μόνο που έχει πάντα την ίδια τιμή (0x8E89BED6). Η χρησιμότητά του είναι διπλή. Προστατεύει από τυχαίες παρεμβολές που μπορούν να έχουν τη μορφή του preamble, ενώ, παράλληλα, δίνει τη δυνατότητα στο δέκτη να αναγνωρίσει την αρχή του πακέτου που περιμένει να λάβει. Το τελευταίο είναι σημαντικό, γιατί ένας δέκτης δεν μπορεί να γνωρίζει επακριβώς, πότε ο πομπός θα ξεκινήσει τη μετάδοση, οπότε, όταν αναμένεται λήψη δεδομένων, πρέπει να ελέγχει συνεχώς τα τελευταία 40 bits που έλαβε, για το αναμενόμενο preamble και access address. Εξαιτίας της χρήσης αυτής, το πρότυπο επιβάλει ορισμένους κανόνες ως προς την ακολουθία bits που απαρτίζουν το access address, το οποίο κατά τα άλλα είναι απλώς ένας τυχαίος αριθμός. Τα preamble και access address εξαιρούνται, επίσης, από τη διαδικασία του data whitening που περιγράψαμε παραπάνω, αφού, λόγω του τρόπου με τον οποίο ορίζονται, δεν περιέχουν μεγάλες ακολουθίες όμοιων bit.

Στο τέλος του πακέτου έχουμε 24 bit cyclic redundancy check (CRC). Το CRC υπολογίζεται σε όλο το μήκος του πακέτου, εκτός των preamble και access address. Με βάση το μέγεθος του πακέτου και του CRC, το τελευταίο έχει τη δυνατότητα ανίχνευσης όλων των περιττών αριθμών σφαλμάτων, καθώς και σφάλματα 2 και 4 bits. Ο shift register που χρησιμοποιείται για τον υπολογισμό του CRC, αρχικοποιείται σε μία τιμή, η οποία αποτελεί παράμετρο της σύνδεσης. Στα advertising πακέτα αρχικοποιείται στην τιμή 0x555555. Το πολυώνυμο που χρησιμοποιείται είναι το ακόλουθο:

$$x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x^1 + x^0$$

2.3.4.2 Header – Advertising πακέτα

Αμέσως πριν τα δεδομένα του ανώτερου επίπεδου, υπάρχει ένα 16-bit header. Το header είναι διαφορετικό για τα δύο είδη πακέτων. Στα advertising πακέτα έχει την ακόλουθη μορφή (LSB to MSB):

PDU Type (4 bits)	**Reserved** (2 bits)	**TxAdd** (1 bits)	**RxAdd** (1 bits)	**Length** (6 bits)	**Reserved** (2 bits)

Υπάρχουν εφτά διαφορετικοί τύποι advertising πακέτων. Καθένας ορίζει μια διαφορετική λειτουργία του πρωτοκόλλου, με τη δική της μορφή για τα δεδομένα που περιέχονται στο πακέτο. Οι τύποι και οι αντίστοιχες λειτουργίες είναι οι εξής:

- **ADV_IND**: Connectable undirected advertising event
- **ADV_DIRECT_IND**: Connectable directed advertising event
- **ADV_NONCONN_IND**: Non-connectable undirected advertising event
- **ADV_SCAN_IND**: Scannable undirected advertising event
- **SCAN_REQ**: Scan request from scanner to advertiser
- **SCAN_RSP**: Response from advertiser to scan request from scanner
- **CONNECT_REQ**: Connection request

Τα TxAdd και RxAdd bits καθορίζουν τον τύπο της διεύθυνσης του αντίστοιχου peer (transmitter/receiver), για τις λειτουργίες όπου αυτά είναι απαραίτητα. Διαφορετικά, δε χρησιμοποιούνται και θεωρούνται επίσης Reserved.

Το μέγιστο μέγεθος δεδομένων σε ένα advertising πακέτο είναι 37 bytes. Τα 6 από αυτά όμως, χρησιμοποιούνται πάντα για την αποστολή της διεύθυνσης του advertiser. Έτσι απομένουν 31 bytes για την αποστολή άλλων δεδομένων.

2.3.4.3 Header – Πακέτα δεδομένων

Στα πακέτα δεδομένων, το header έχει την ακόλουθη μορφή (LSB to MSB):

LLID (2 bits)	**NESN** (1 bits)	**SN** (1 bits)	**MD** (1 bits)	**Reserved** (3 bits)	**Length** (5 bits)	**Reserved** (3 bits)

Το Logical Link Identifier (LLID) καθορίζει τον τύπο του πακέτου δεδομένων και το περιεχόμενο του payload. Υπάρχουν τρεις επιτρεπτές τιμές.

LLID	**Είδος πακέτου**
00b	Reserved
01b	LL Data PDU Συνέχεια ενός διασπασμένου μηνύματος L2CAP ή άδεια PDU.
10b	LL Data PDU Εκκίνηση ενός μηνύματος L2CAP ή ολοκληρωμένο μήνυμα L2CAP χωρίς διάσπαση.
11b	LL Control PDU Χρησιμοποιείται για την επικοινωνία μεταξύ των Link Layers διαφορετικών συσκευών.

Όπως παρατηρούμε, δεν υπάρχει LLID που να δηλώνει το τέλος ενός μηνύματος L2CAP. Ένα L2CAP μήνυμα θεωρείται ότι ολοκληρώνεται, όταν γίνεται η εκκίνηση του επόμενου. Αυτό επιτρέπει την αποστολή ενός ή περισσότερων άδειων πακέτων, χωρίς να επηρεάζεται η λειτουργία του L2CAP. Τα άδεια πακέτα χρησιμεύουν στις περιπτώσεις εκείνες, στις οποίες μια συσκευή πρέπει να στείλει ένα πακέτο (π.χ. acknowledgement, εκκίνηση connection event από τον master), αλλά δεν έχει διαθέσιμα δεδομένα προς αποστολή.

Η επικοινωνία μεταξύ των Link Layers διαφορετικών συσκευών επιτυγχάνεται με τη χρήση των LL Control PDUs. Ένα LL Control PDU αποτελείται από 1 byte opcode και ως 26 bytes δεδομένων. Το πρότυπο ορίζει μια σειρά από requests και τα αντίστοιχα responses, καθώς και indications, τα οποία μπορεί να χρησιμοποιήσει το επίπεδο ζεύξης δεδομένων,

προκειμένου να φέρει εις πέρας τις λειτουργίες που ορίζονται για αυτό. Ορισμένα από τα requests μπορούν να σταλούν μόνο από τον ένα peer (master ή slave).

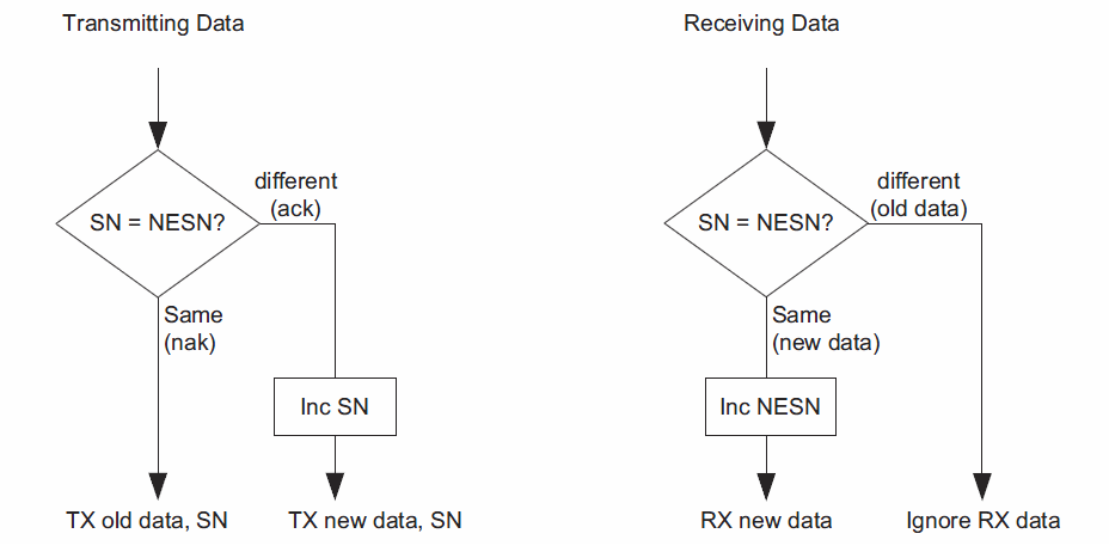
Value	**Opcode**
0x00	LL_CONNECTION_UPDATE_REQ (master only) Αλλαγή παραμέτρων της σύνδεσης.
0x01	LL_CHANNEL_MAP_REQ (master only) Ανανέωση του channel map του AFH.
0x02	LL_TERMINATE_IND Τερματισμός σύνδεσης.
0x03	LL_ENC_REQ (master only) Αίτημα για κρυπτογράφηση της σύνδεσης. Περιέχει τα απαραίτητα δεδομένα από την πλευρά του master.
0x04	LL_ENC_RSP (slave only) Απάντηση στο αίτημα για κρυπτογράφηση της σύνδεσης. Περιέχει τα απαραίτητα δεδομένα από την πλευρά του slave.
0x05	LL_START_ENC_REQ (slave only) Εκκίνηση κρυπτογράφησης της σύνδεσης.
0x06	LL_START_ENC_RSP Επιβεβαίωση από τους peers για την εκκίνηση κρυπτογράφησης.
0x07	LL_UNKNOWN_RSP Άγνωστη εντολή.
0x08	LL_FEATURE_REQ (master only) Ανταλλαγή υποστηριζόμενων από το link layer χαρακτηριστικών. Το μήνυμα περιέχει τα χαρακτηριστικά που υποστηρίζει ο master.
0x09	LL_FEATURE_RSP Ανταλλαγή υποστηριζόμενων από το link layer χαρακτηριστικών. Το μήνυμα περιέχει τα κοινά χαρακτηριστικά που θα χρησιμοποιηθούν.
0x0A	LL_PAUSE_ENC_REQ (master only) Αίτημα για προσωρινή διακοπή της κρυπτογράφησης. Χρησιμοποιείται, για παράδειγμα, όταν μια συσκευή θέλει να αλλάξει το τρέχον κλειδί κρυπτογράφησης με ένα νέο.
0x0B	LL_PAUSE_ENC_RSP Επιβεβαίωση από τους peers για την προσωρινή διακοπή της κρυπτογράφησης.
0x0C	LL_VERSION_IND Ανταλλαγή πληροφοριών έκδοσης των συσκευών.
0x0D	LL_REJECT_IND Απόρριψη εντολής. Το μήνυμα περιέχει τον κωδικό λάθους.
0x0E	LL_SLAVE_FEATURE_REQ (slave only, new in 4.1) Ανταλλαγή υποστηριζόμενων από το link layer χαρακτηριστικών. Το μήνυμα περιέχει τα χαρακτηριστικά που υποστηρίζει ο slave.
0x0F	LL_CONNECTION_PARAM_REQ (new in 4.1) Αίτημα για αλλαγή των παραμέτρων της σύνδεσης. Το μήνυμα περιέχει τις νέες επιθυμητές τιμές.
0x10	LL_CONNECTION_PARAM_RSP (new in 4.1) Απάντηση στο αίτημα για αλλαγή των παραμέτρων της σύνδεσης. Το μήνυμα περιέχει εναλλακτικές στις προτεινόμενες ρυθμίσεις.
0x11	LL_REJECT_IND_EXT (new in 4.1) Απόρριψη εντολής (extension). Το μήνυμα περιέχει, επιπλέον, τον κώδικα της εντολής που απορρίπτεται.
0x12	LL_PING_REQ (new in 4.1) Αποστολή ping για τον έλεγχο παρουσίας του peer link layer.

0x13	LL_PING_RSP (new in 4.1) Απάντηση στο ping από το peer link layer.
0x14 – 0xFF	Reserved

Το bit More Data (MD) καθορίζει αν το τρέχον connection event πρέπει να συνεχιστεί ή να τερματιστεί. Η λειτουργία του περιγράφεται παρακάτω, όπου και ασχολούμαστε με τις λεπτομέρειες της σύνδεσης μεταξύ συσκευών.

Τα Sequence Number (SN) και Next Expected Sequence Number (NESN) bits αποτελούν τον μηχανισμό acknowledgement και flow control του BLE. Σε κάθε πακέτο, το SN ορίζει το sequence number του συγκεκριμένου πακέτου, ενώ το NESN ορίζει το sequence number του επόμενου πακέτου που ο αποστολέας αναμένει να λάβει. Εφόσον το sequence number είναι 1 bit, σημαίνει ότι, κάθε στιγμή, μόνο ένα πακέτο μπορεί να είναι μη επιβεβαιωμένο (acknowledged) σε κάθε peer. Μια συσκευή δεν μπορεί να στείλει νέο πακέτο πριν επιβεβαιωθεί το προηγούμενο που έστειλε.

Το bit NESN είναι εκείνο που πραγματοποιεί την επιβεβαίωση των πακέτων. Ένας peer επιβεβαιώνει ένα πακέτο, όταν, στο επόμενο πακέτο που ο ίδιος στέλνει, έχει αυξήσει (ουσιαστικά αντιστρέψει, αφού πρόκειται για 1 bit) το NESN. Ένας peer που λαμβάνει ένα πακέτο, μπορεί, ελέγχοντας το NESN, να δει αν το προηγούμενο πακέτο που έστειλε έχει επιβεβαιωθεί. Αν το NESN είναι διαφορετικό από το SN του πακέτου που έστειλε, τότε το προηγούμενο πακέτο θεωρείται επιβεβαιωμένο και μπορεί να σταλεί νέο. Διαφορετικά, το προηγούμενο πακέτο θεωρείται χαμένο και ξαναστέλνεται. Παράλληλα, ο λήπτης μπορεί να ελέγξει αν το πακέτο που έλαβε είναι καινούριο ή επαναμετάδοση πακέτου, το οποίο έχει ήδη ληφθεί. Αυτό γίνεται με τη σύγκριση του SN του εισερχόμενου πακέτου με το sequence number που ο λήπτης αναμένει να λάβει. Το τελευταίο ανανεώνεται με τη λήψη κάθε πακέτου. Αν το SN του πακέτου συμπίπτει με το αναμενόμενο, τότε το πακέτο θεωρείται νέο και προχωρά η λήψη του. Διαφορετικά, θεωρείται παλιό και απορρίπτεται.



Το BLE, σε αντίθεση με το Bluetooth Classic, όπου τα πακέτα επιβεβαιώνονται άμεσα, ακολουθεί μια τακτική lazy acknowledgement. Σύμφωνα με αυτή, μία συσκευή δεν είναι απαραίτητο να επιβεβαιώσει ένα πακέτο αμέσως μόλις το λάβει, αν εκείνη τη στιγμή δεν έχει δεδομένα προς αποστολή. Μπορεί να περιμένει και να στείλει την επιβεβαίωση στο επόμενο πακέτο δεδομένων. Έτσι αποφεύγεται η άσκοπη αποστολή κενών πακέτων και η σπατάλη ενέργειας.

Το μέγιστο μέγεθος δεδομένων σε ένα data πακέτο είναι 31 bytes. Όμως, όταν ενεργοποιείται η κρυπτογράφηση, τα τέσσερα τελευταία bytes πριν το CRC, χρησιμοποιούνται για την αποστολή του Message Integrity Check (MIC), με το οποίο πραγματοποιείται πιστοποίηση αυθεντικότητας στο κρυπτογραφημένο πακέτο. Η ύπαρξη του MIC σημαίνει ότι το μέγιστο μέγεθος δεδομένων για κρυπτογραφημένα πακέτα είναι: 31 – 4 = 27 bytes. Όμως, προκειμένου να μειωθεί η πολυπλοκότητα του Link Layer, δεν επιτρέπεται ακόμη και σε μη κρυπτογραφημένα πακέτα να έχουν μέγεθος μεγαλύτερο από 27 bytes, οπότε, αυτό είναι το τελικό μέγιστο ωφέλιμο φορτίο, το οποίο παρέχεται στα ανώτερα επίπεδα για μεταφορά δεδομένων.

2.3.5 Inter Frame Space – Throughput

Το Inter Frame Space (T_IFS) είναι ένα μέγεθος, που ορίζεται στο πρότυπο του BLE, το οποίο αντιστοιχεί στον χρόνο που μεσολαβεί μεταξύ διαδοχικών πακέτων που στέλνονται ή λαμβάνονται. Δηλαδή, μεταξύ της αποστολής ή λήψης του τελευταίου bit ενός πακέτου, και αυτής του πρώτου bit του επόμενου, υπάρχει συγκεκριμένος χρόνος αναμονής, ο οποίος είναι 150 μs. Το Inter Frame Space χρησιμοποιείται σε πολλά σημεία της λειτουργίας του BLE. Για παράδειγμα, στα advertising και connection events, τα οποία περιγράφουμε παρακάτω, η συσκευή που αναμένει να λάβει δεδομένα, ελέγχει το μέσο 150μs μετά το τέλος της αποστολής των δικών της δεδομένων, με μία ανοχή ±2μs. Αν καμία απομακρυσμένη συσκευή δε στείλει δεδομένα σε αυτή τη χρονική στιγμή, τότε θεωρείται ότι δεν πρόκειται να ληφθούν δεδομένα στο τρέχον event.

Όπως αναφέραμε παραπάνω, τα κυκλώματα αποστολής και λήψης των BLE συσκευών δεν πρέπει να λειτουργούν συνεχώς για μεγάλο χρονικό διάστημα, διότι αυξάνεται η θερμοκρασία τους, επηρεάζοντας τις ιδιότητές τους. Το Inter Frame Space επιτρέπει στα κυκλώματα αυτά να κρυώσουν, μετά την ολοκλήρωση της τελευταίας λειτουργίας τους.

Με βάση το Inter Frame Space και τα μεγέθη των πακέτων που στέλνονται μεταξύ των δύο πλευρών μιας σύνδεσης, είναι δυνατόν να υπολογιστεί το μέγιστο throughput δεδομένων που μπορούν να σταλούν προς μία κατεύθυνση. Έστω ότι μια συσκευή χρησιμοποιεί το μέγιστο δυνατό μήκος δεδομένων, δηλαδή 27 bytes, και στέλνει κρυπτογραφημένα πακέτα στη μέγιστη ροή που επιτρέπεται από το πρότυπο. Κάθε πακέτο έχει μήκος 328 bits και πρέπει να επιβεβαιώνεται από τη δεύτερη συσκευή με ένα κενό πακέτο των 80 bits. Επίσης, ανάμεσα στα πακέτα πρέπει να μεσολαβεί χρόνος ίσος με T_IFS. Κάθε bit στέλνεται σε 1 μs, οπότε, για την αποστολή 27 bytes δεδομένων, απαιτούνται:

$$328 + 150 + 80 + 150 = 708\ \mu s$$

Το μέγιστο throughput, λοιπόν, που προκύπτει είναι:

$$\frac{1000000}{708} \times 27 \times 8 \approx 305 kbps$$

2.3.6 Advertising – Scanning

Η διαδικασία του advertising πραγματοποιείται μέσω της περιοδικής εκτέλεσης advertising events από μια συσκευή advertiser. Σε κάθε advertising event, ο advertiser στέλνει το ίδιο πακέτο σε ένα ή περισσότερα από τα τρία κανάλια advertising. Η ρύθμιση αυτή γίνεται από τον host στον controller μέσω κατάλληλης εντολής του HCI. Ο χρόνος που μεσολαβεί, ανάμεσα σε διαδοχικά advertising events, ρυθμίζεται επίσης από τον host και ονομάζεται advertising interval. Οι δυνατές τιμές του advertising interval είναι από 20 ms μέχρι 10.24 sec, με ανάλυση 0.625 ms. Η ελάχιστη τιμή του advertising interval γίνεται 100 ms, στην περίπτωση των scannable και non-connectable undirected advertising events. Στο advertising interval προστίθεται κάθε φορά μια τυχαία τιμή έως 10 ms. Αυτό γίνεται για την αποφυγή ενδεχόμενου συγχρονισμού μεταξύ advertising events διαφορετικών συσκευών, που θα είχε ως αποτέλεσμα καμία από τις δύο να μην μπορεί να στείλει δεδομένα.

Αν κάτι τέτοιο προβλέπεται από τον τύπο του event, τότε, μετά την αποστολή ενός πακέτου advertising, και αφού περιμένει χρόνο ίσο με το T_IFS, ο advertiser ελέγχει το μέσο για τυχόν scan ή connection requests (τύποι πακέτων SCAN_REQ και CONNECT_REQ αντίστοιχα) από απομακρυσμένες συσκευές. Αν λάβει ένα πακέτο τύπου SCAN_REQ, τότε απαντά με ένα πακέτο τύπου SCAN_RSP, παρέχοντας επιπλέον δεδομένα. Σε αυτήν την περίπτωση, το advertising event συνεχίζεται. Αν λάβει ένα πακέτο τύπου CONNECT_REQ, τότε το advertising event τερματίζεται, και εκκινείται η διαδικασία σύνδεσης με την απομακρυσμένη συσκευή, την οποία περιγράφουμε παρακάτω.

Ο τύπος του advertising event προκύπτει από το πακέτο που στέλνεται από τον advertiser, και, συγκεκριμένα από τα PDU Type bits του advertising packet header. Υπάρχουν 4 τύποι advertising events:

- **Connectable undirected advertising event**: Ο advertiser στέλνει advertising πακέτα τύπου ADV_IND και, στη συνέχεια, ελέγχει το μέσο για τη λήψη scan ή connection requests από κάποιον scanner ή initiator. Είναι η γενική διαδικασία advertising που εκτελείται από BLE συσκευές, προκειμένου να κάνουν γνωστή την παρουσία τους σε ενδιαφερόμενους peers.
- **Connectable directed advertising event**: Ο advertiser στέλνει advertising πακέτα τύπου ADV_DIRECT_IND και, στη συνέχεια, ελέγχει το μέσο για τη λήψη connection request από μια συγκεκριμένη συσκευή, της οποίας η διεύθυνση περιλαμβάνεται στα δεδομένα του πακέτου. Η διαδικασία αυτή χρησιμοποιείται, όταν μια συσκευή θέλει να στείλει δεδομένα γρήγορα σε μια γνωστή συσκευή, για την οποία, η πιθανότητα να βρίσκεται εκείνη τη στιγμή σε εμβέλεια και να περιμένει δεδομένα είναι μεγάλη. Σε αυτόν τον τύπο advertising, το interval τίθεται σε μικρή τιμή, ώστε να υπάρχει χαμηλό latency στη λήψη των δεδομένων, χωρίς η δεύτερη συσκευή να χρειάζεται να ελέγχει συνεχώς το μέσο.
- **Non-connectable undirected advertising event**: Ο advertiser στέλνει advertising πακέτα τύπου ADV_NONCONN_IND, χωρίς, στη συνέχεια, να ελέγχει το μέσο. Δε δέχεται δηλαδή scan ή connection requests. Η διαδικασία αυτή χρησιμοποιείται από συσκευές που απλώς μεταδίδουν δεδομένα περιοδικά, για ενδιαφερόμενες συσκευές (broadcast), οπότε δεν απαιτείται σύνδεση ή αποστολή επιπλέον δεδομένων.
- **Scannable undirected advertising event**: Ο advertiser στέλνει advertising πακέτα τύπου ADV_SCAN_IND και, στη συνέχεια, ελέγχει το μέσο για τη λήψη scan request από κάποιον scanner. Η διαδικασία αυτή είναι παρόμοια με την προηγούμενη. Χρησιμοποιείται, δηλαδή, από συσκευές που απλώς μεταδίδουν δεδομένα περιοδικά, με τη δυνατότητα, όμως, αποστολής επιπλέον δεδομένων στις ενδιαφερόμενες συσκευές, μέσω της αποστολής SCAN_RSP ως απάντηση σε SCAN_REQ που στέλνονται από αυτές.

Τα πακέτα τύπου SCAN_REQ και ADV_DIRECT_IND, περιέχουν στα δεδομένα τους μόνο τις διευθύνσεις των δύο συσκευών. Τα πακέτα τύπου SCAN_RSP, ADV_IND, ADV_NONCONN_IND και ADV_SCAN_IND, εκτός από τη διεύθυνση του advertiser, μπορούν να περιέχουν και μια σειρά από δεδομένα. Η μορφή των δεδομένων αυτών ορίζεται στο Generic Access Profile (GAP). Τα δεδομένα αυτά, με μέγιστο μέγεθος 31 bytes, αποτελούνται από μια ακολουθία από AD (Advertising Data) structures. Κάθε AD structure ξεκινά με ένα byte που ορίζει το μέγεθος των δεδομένων που ακολουθούν. Τα δεδομένα αυτά ξεκινούν με ένα byte που δηλώνει τον τύπο του AD structure (AD type), το οποίο ακολουθείται από τα δεδομένα του (AD data). Με αυτόν τον τρόπο, μια συσκευή που δεν αναγνωρίζει τον τύπο κάποιων από τα AD structures που περιλαμβάνονται σε ένα πακέτο, μπορεί απλώς να τα προσπεράσει, χρησιμοποιώντας το μήκος που βρίσκεται πάντα στο πρώτο byte. Το πρότυπο ορίζει μια σειρά από AD types και τη μορφή των αντίστοιχων AD data. Ο ακόλουθος πίνακας περιέχει μερικά ενδεικτικά AD types.

AD type	**Data Type**	**Περιγραφή**
0x01	«Flags»	Χρησιμοποιούνται για να δηλώσουν συγκεκριμένες ιδιότητες του advertiser, όπως το GAP Discoverable Mode στο οποίο βρίσκεται (limited ή general) και η υποστήριξη του BR/EDR.
0x02 0x03 0x06 0x07	«Incomplete List of 16-bit Service Class UUIDs» «Complete List of 16-bit Service Class UUIDs» «Incomplete List of 128-bit Service Class UUIDs» «Complete List of 128-bit Service Class UUIDs»	Λίστα με τις υπηρεσίες που υποστηρίζονται από τον advertiser. Η λίστα μπορεί να είναι πλήρης ή να περιέχει μέρος μόνο των υποστηριζόμενων υπηρεσιών.
0x08 0x09	«Shortened Local Name» «Complete Local Name»	Το GAP Device Name της συσκευής, ολόκληρο ή κομμένο.
0x0A	«Tx Power Level»	Η ισχύς εκπομπής σε dBm (1 byte). Μπορεί να χρησιμοποιηθεί, σε συνδυασμό με το RSSI, για τον κατά προσέγγιση υπολογισμό της απόστασης από τη συσκευή.
0x14 0x15	«List of 16-bit Service Solicitation UUIDs» «List of 128-bit Service Solicitation UUIDs»	Χρησιμοποιείται από μια συσκευή, προκειμένου να δηλώσει ότι υποστηρίζει τις συγκεκριμένες υπηρεσίες ως client. Άλλες συσκευές μπορούν να ελέγξουν αυτά τα δεδομένα για να αποφασίσουν αν θέλουν να συνδεθούν με αυτήν.
0x16 0x21	«Service Data - 16-bit UUID» «Service Data - 128-bit UUID»	Χρησιμοποιείται από μια συσκευή, προκειμένου να μεταδίδει δεδομένα κάποιας υπηρεσίας που περιλαμβάνεται σε αυτήν, χωρίς να απαιτείται σύνδεση.
0xFF	«Manufacturer Specific Data»	Custom δεδομένα που ορίζονται από κάποιον κατασκευαστή. Τα δεδομένα ξεκινούν με το 16-bit αναγνωριστικό του κατασκευαστή. Το format των υπόλοιπων δεδομένων καθορίζεται από αυτόν.

Η διαδικασία του scanning πραγματοποιείται μέσω του περιοδικού ελέγχου των advertising καναλιών από τον scanner, για ενδεχόμενα advertising πακέτα από advertisers. Ο περιοδικός έλεγχος ρυθμίζεται από τον host με βάση δύο μεγέθη, τα scan interval και scan window. Οι δυνατές τιμές των δύο μεγεθών είναι από 2.5 ms έως 10.24 sec με ανάλυση 0.625 ms. Το scan window πρέπει να είναι μικρότερο ή ίσο του scan interval. Ο έλεγχος ξεκινά ανά χρόνο ίσο με το scan interval, σε διαφορετικό advertising κανάλι κάθε φορά, και εκτελείται

για χρόνο ίσο με το scan window. Μέσω των scan interval και scan window μπορεί να οριστεί το duty cycle του scanning. Αν τα δύο μεγέθη είναι ίσα, έχουμε 100% duty cycle. Υπάρχουν δύο τύποι scanning, το active και το passive. Κατά το passive scanning, ο scanner δε στέλνει πακέτα. Αντίθετα, κατά το active scanning, και στην περίπτωση που ο scanner λάβει ένα advertising πακέτο τύπου ADV_IND ή ADV_SCAN_IND, στέλνει ένα πακέτο τύπου SCAN_REQ στον advertiser και περιμένει να λάβει το SCAN_RSP από αυτόν.

Παρόμοια διαδικασία εκτελείται και από έναν initiator. Ο initiator ελέγχει τα advertising κανάλια με τον ίδιο τρόπο όπως και ο scanner. Η διαφορά είναι ότι δε στέλνει SCAN_REQ, αλλά, αν λάβει ένα advertising πακέτο τύπου ADV_IND ή ADV_DIRECT_IND (με τη διεύθυνσή του), τότε, στέλνει ένα πακέτο CONNECT_REQ, προκειμένου να συνδεθεί με τον advertiser.

2.3.7 Σύνδεση

Μετά την αποστολή του πακέτου CONNECT_REQ από τον initiator και τη λήψη του από τον advertiser, ξεκινά η σύνδεση των δύο συσκευών, με τον initiator να γίνεται master και τον advertiser να γίνεται slave. Το πακέτο CONNECT_REQ περιέχει τις παραμέτρους της σύνδεσης, όπως ορίζονται από τον master. Αυτές είναι οι εξής:

- Το Access Address που θα χρησιμοποιηθεί στη συγκεκριμένη σύνδεση. Είναι μια τυχαία τιμή, που πρέπει όμως να ικανοποιεί ορισμένες προϋποθέσεις, που ορίζονται στο πρότυπο. Για παράδειγμα, εφόσον εξαιρείται από το data whitening, δεν πρέπει να περιέχει περισσότερα από έξι όμοια bits στη σειρά.

- Η τιμή αρχικοποίησης του linear feedback shift register (LFSR) για τον υπολογισμό του CRC. Είναι μια τυχαία τιμή.

- Η τιμή του hop που χρησιμοποιείται από το AFH για τον υπολογισμό του επόμενου καναλιού. Είναι μια τυχαία τιμή από 5 ως 16.

- Τα κανάλια (channel map) που θα χρησιμοποιηθούν από το AFH. Ο master έχει τη δυνατότητα να απενεργοποιήσει συγκεκριμένα κανάλια. Αυτό μπορεί να χρησιμεύσει, για παράδειγμα, αν σε αυτά τα κανάλια υπάρχουν μεγάλες παρεμβολές.

- Το connection interval της σύνδεσης, το οποίο καθορίζει τον χρόνο που μεσολαβεί μεταξύ διαδοχικών connection events. Οι δυνατές τιμές του connection interval είναι από 7.5 ms ως 4 sec, με ανάλυση 1.25 ms. Τα connection events και οι διαδικασίες που ορίζονται για αυτά περιγράφονται παρακάτω.

- Το supervision timeout, το οποίο καθορίζει το μέγιστο χρονικό διάστημα, στο οποίο δεν έχει ληφθεί κάποιο πακέτο από την άλλη συσκευή της σύνδεσης. Αν εκπνεύσει το supervision timeout, η σύνδεση θεωρείται ότι έχει διακοπεί. Οι δυνατές τιμές του supervision timeout είναι από 100 ms ως 32 sec, με ανάλυση 10 ms.

- Το slave latency, το οποίο καθορίζει τον αριθμό των connection events, τα οποία μπορεί να παραλείψει ο slave. Το slave latency δίνει τη δυνατότητα στον slave να μη χρειάζεται να επικοινωνεί με τον master σε κάθε connection event. Οι δυνατές τιμές του είναι από μηδέν, το οποίο σημαίνει ότι ο slave πρέπει να συμμετέχει σε κάθε connection event, μέχρι 500. Σε κάθε περίπτωση όμως, το slave latency πρέπει να έχει τέτοια τιμή, ώστε να μην προκαλεί τη λήξη του supervision timeout.

- Οι τιμές των μεγεθών transmitWindowOffset και transmitWindowSize. Αυτές χρησιμοποιούνται, προκειμένου ο master να καθορίσει πότε θα γίνει η πρώτη αποστολή δεδομένων. Συγκεκριμένα, αυτή θα ξεκινήσει μέσα στο χρονικό διάστημα, που αρχίζει (transmitWindowOffset + 1.25) ms μετά από την αποστολή του CONNECT_REQ, και διαρκεί transmitWindowSize. Αυτή η ρύθμιση δίνει στον master τη δυνατότητα να συντονίσει τις διαφορετικές συνδέσεις, στις οποίες μπορεί να συμμετέχει. Οι δυνατές τιμές του transmitWindowOffset είναι μεταξύ 0 ms και της τιμής του connection interval, με ανάλυση 1.25 ms. Οι δυνατές τιμές του transmitWindowSize είναι από 1.25 ms μέχρι το ελάχιστο μεταξύ των 10 ms και (connection interval - 1.25) ms, πάλι με ανάλυση 1.25 ms. Παρατηρούμε ότι υπάρχει πάντα μια καθυστέρηση τουλάχιστον 1.25 ms, από την αποστολή του CONNECT_REQ, μέχρι την πρώτη επικοινωνία. Αυτή δίνει τη δυνατότητα στους hosts των συσκευών να έχουν έτοιμα δεδομένα προς αποστολή κατά το πρώτο connection event. Επίσης, χρησιμεύει για να επιτρέψει στις συσκευές να επανέλθουν σε κανονική κατάσταση μετά την, πιθανώς εξαντλητική, διαδικασία της σύνδεσης.

- Την τιμή του Sleep Clock Accuracy, το οποίο καθορίζει την ακρίβεια, σε parts per million (ppm), του ρολογιού του master, που χρησιμοποιείται για τη μέτρηση του connection interval. Η τιμή αυτή συνδυάζεται με την αντίστοιχη του slave και από αυτές προκύπτει πόσο θα πρέπει να αυξήσει ο slave το χρόνο ελέγχου του μέσου για connection events (window widening), ώστε να αντιμετωπίσει πιθανές διαφορές στη μέτρηση του χρόνου μεταξύ αυτού και του master.

Ορισμένες από τις παραπάνω παραμέτρους, μπορούν να αλλάξουν και κατά τη διάρκεια της σύνδεσης, με αποστολή κατάλληλων μηνυμάτων. Οι παράμετροι αυτές είναι το connection interval, το supervision timeout, το slave latency και το channel map. Ο master είναι αυτός που ελέγχει τις παραμέτρους της σύνδεσης. Ο slave μπορεί να ζητήσει από τον master συγκεκριμένες τιμές για τις παραμέτρους, αλλά δεν είναι απαραίτητο ότι ο master θα τις δεχτεί.

Μετά την αποστολή του CONNECT_REQ, ο initiator περνάει άμεσα σε Connection State ως master. Αντίστοιχα, ο advertiser, μετά τη λήψη του CONNECT_REQ, περνάει άμεσα σε Connection State ως slave. Δηλαδή, και οι δύο συσκευές θεωρούν ότι είναι συνδεμένες, χωρίς να υπάρχει κάποια επιβεβαίωση. Το πρότυπο ορίζει ότι, σε αυτή την περίπτωση, η σύνδεση είναι μεν δημιουργημένη, αλλά όχι εγκατεστημένη (established). Θεωρείται εγκατεστημένη μόνο μετά την αποστολή των πρώτων δεδομένων από κάθε πλευρά. Η μόνη διαφορά που υπάρχει μεταξύ των δύο περιπτώσεων είναι στο supervision timeout. Αυτό, αρχικά, τίθεται σε τιμή ίση με έξι φορές το connection interval, ενώ, μετά την εγκατάσταση της σύνδεσης, τίθεται στην τιμή που όρισε ο master. Αυτό επιτρέπει τη γρήγορη διακοπή συνδέσεων, που δεν ολοκληρώνονται σε σύντομο χρονικό διάστημα από τη στιγμή που δημιουργούνται.

Από τη στιγμή που θα ξεκινήσει η σύνδεση, οι δύο συσκευές μπορούν πλέον να επικοινωνούν στα κανάλια δεδομένων, μέσω της περιοδικής διαδικασίας των connection events. Ένα connection event εκκινείται από τον master με την αποστολή ενός, πιθανώς άδειου, πακέτου, ανά χρονικό διάστημα ίσο με το connection interval, και σε διαφορετικό κανάλι κάθε φορά, το οποίο προκύπτει από το AFH. Αν το slave latency δεν είναι μηδενικό, ο slave έχει τη δυνατότητα να αγνοήσει ένα αριθμό από connection events, ίσο με το slave latency. Για να συμμετέχει στο connection event, ο slave, αφού λάβει το πακέτο του master και περιμένει για χρόνο T_IFS, στέλνει το δικό του πακέτο στο ίδιο κανάλι. Αντίστοιχα, ο master, αφού λάβει το πακέτο του slave, μπορεί με τη σειρά του να στείλει νέο πακέτο. Η επικοινωνία συνεχίζεται με τον ίδιο τρόπο και στο ίδιο κανάλι μέχρι τον τερματισμό του connection event. Ο τερματισμός του connection event γίνεται μέσω του bit More Data (MD) του header των πακέτων δεδομένων. Αν καμία από τις συσκευές δεν έχει θέσει το bit MD, τότε το connection event θεωρείται λήξαν. Αν κάποια ή και οι δύο έχουν θέσει το bit MD,

τότε το connection event συνεχίζεται. Δεν είναι όπως υποχρεωτικό για τις συσκευές να συμμετέχουν σε αυτό. Αν κάποια από τις συσκευές δε στείλει πακέτο σε χρόνο T_IFS μετά την αποστολή του τελευταίου από την άλλη συσκευή, το connection event τερματίζεται.

Η σύνδεση μεταξύ των συσκευών παραμένει ενεργή μέχρι, είτε να συμβεί κάποιο σφάλμα, όπως η λήξη του supervision timeout, είτε μία από αυτές να κάνει αποσύνδεση, στέλνοντας στην άλλη ένα LL Control PDU τύπου LL_TERMINATE_IND.

2.4 Host Controller Interface (HCI)

Το Host Controller Interface (HCI) είναι η προτυποποιημένη διεπαφή μεταξύ του host και του controller μιας BLE συσκευής, μέσω της οποίας γίνεται η επικοινωνία των δύο τμημάτων του συστήματος, προκειμένου να εκτελεστούν λειτουργίες του BLE και να γίνει ανταλλαγή δεδομένων, όπως προβλέπεται από το πρότυπο.

Κατά τη συνοπτική περιγραφή της στοίβας πρωτοκόλλων του BLE, κάναμε μια γενική αναφορά στον ρόλο του HCI στην αρχιτεκτονική του BLE, και πώς αυτό μπορεί να χρησιμοποιηθεί για την υποστήριξη διαφορετικών αρχιτεκτονικών στις BLE συσκευές. Όπως έχουμε ήδη αναφέρει, το πρότυπο του BLE ορίζει μια σειρά από διαφορετικά transport layers, πάνω από τα οποία μπορεί να πραγματοποιηθεί η επικοινωνία του HCI. Τα transport layers που ορίζονται είναι τα UART, USB, Secure Digital (SD) και 3-wire UART. Σε BLE συσκευές, οι οποίες ακολουθούν την αρχιτεκτονική του ενός chipset, δεν είναι απαραίτητο να υπάρχει το HCI ως physical transport, αφού η επικοινωνία μεταξύ host και controller μπορεί να υλοποιηθεί εσωτερικά, με τρόπο που ορίζει ο κατασκευαστής της συσκευής.

Στο HCI ορίζονται τρεις τύποι πακέτων. Επίσης, για κάθε σύνδεση μεταξύ συσκευών, δημιουργείται από το HCI ένα κανάλι, το οποίο χρησιμοποιείται για την αναγνώριση των δεδομένων που ανταλλάσσονται. Οι τύποι πακέτων είναι οι εξής:

- **Command:** Στέλνονται από τον host στον controller, προκειμένου ο πρώτος να ελέγξει ή να ρυθμίσει γενικές λειτουργίες του δεύτερου, να ζητήσει την εκτέλεση κάποιας συγκεκριμένης διαδικασίας, και να χειριστεί τις ενεργές συνδέσεις στις οποίες συμμετέχει, αλλάζοντας παραμέτρους και εκτελώντας λειτουργίες του BLE. Η δομή του πακέτου είναι η ακόλουθη:

Opcode (2 bytes)	**Parameters Length** (1 byte)	**Parameters**

- **Event:** Στέλνονται από τον controller στον host, για την ενημέρωση του δεύτερου για γεγονότα που σχετίζονται με τη λειτουργία του controller ή με διαδικασίες του BLE. Τα περισσότερα από τα events, που ορίζονται στο πρότυπο, είναι απαντήσεις σε αντίστοιχες εντολές που στέλνονται από τον host. Η δομή του πακέτου είναι η ακόλουθη:

Event Code (1 byte)	**Parameters Length** (1 byte)	**Parameters**

- **Data:** Τα πακέτα αυτά περιέχουν δεδομένα που ανταλλάσσονται μεταξύ συσκευών, οι οποίες είναι συνδεμένες μέσω του BLE. Στέλνονται και από τις δυο πλευρές, από τον host, όταν θέλει να στείλει δεδομένα στην απομακρυσμένη συσκευή, και από το controller, όταν λαμβάνονται δεδομένα από αυτή. Η δομή του πακέτου είναι η ακόλουθη:

Handle/Flags (2 bytes)	**Data Length** (1 byte)	**Data**

Εφόσον το HCI ορίζει τον τρόπο επικοινωνίας μεταξύ των host και controller, περιλαμβάνει εντολές και γεγονότα για κάθε λειτουργία, διαδικασία ή ρύθμιση, που προβλέπεται από το πρότυπο του BLE για συμβατές με αυτό συσκευές. Ως εκ τούτου, το HCI είναι αρκετά εκτεταμένο. Το BLE χρησιμοποιεί κάποιες κοινές εντολές και γεγονότα με το Bluetooth Classic, ορίζοντας επιπλέον τύπους, για τις ξεχωριστές του λειτουργίες. Στον ακόλουθο πίνακα περιέχονται, ενδεικτικά, ορισμένες εντολές και γεγονότα του HCI, χωρισμένα σε ομάδες, που ορίζονται στο πρότυπο.

Group	Commands / Events
Generic Events	Command Complete Event Command Status Event Hardware Error Event
Device Setup	Reset Command
Controller Information	Read Local Version Information Command Read Local Supported Commands Command Read Local Supported Features Command Read BDADDR Command LE Read Local Supported Features Command LE Read Supported States Command
Remote Information	Read Remote Version Information Command Read Remote Version Information Complete Event LE Read Remote Used Features Command LE Read Remote Used Features Complete Event
Controller Configuration	LE Set Advertise Enable Command LE Set Advertising Data Command LE Set Advertising Parameters Command LE Set Random Address Command LE Set Scan Response Data Command
Device Discovery	LE Advertising Report Event LE Set Scan Enable Command LE Set Scan Parameters Command
Connection Setup	LE Create Connection Command LE Create Connection Cancel Command LE Connection Complete Event Disconnect Command Command Disconnection Complete Event

Connection State	LE Connection Update Command LE Connection Update Complete Event New in 4.1: LE Remote Connection Parameter Request Reply Command LE Remote Connection Parameter Request Negative Reply Command LE Remote Connection Parameter Request Event
Authentication Encryption	Encryption Change Event Encryption Key Refresh Complete Event LE Encrypt Command LE Long Term Key Requested Event LE Long Term Key Requested Reply Command LE Long Term Key Requested Negative Reply Command LE Rand Command LE Start Encryption Command
Link Information	Read Transmit Power Level Command Read RSSI Command LE Read Advertising Channel TX Power Command LE Read Channel Map Command
Test	LE Receiver Test Command LE Transmitter Test Command LE Test End Command

Αξίζει εδώ να επισημάνουμε την εντολή LE_Encrypt, με την οποία ο host μπορεί να ζητήσει από τον controller να κρυπτογραφήσει κάποια δεδομένα. Η ύπαρξη αυτής της εντολής αποτελεί ακόμα μια περίπτωση, από την οποία φαίνεται η προσήλωση του προτύπου στην ελαχιστοποίηση της κατανάλωσης ενέργειας. Το πρότυπο δίνει τη δυνατότητα κρυπτογραφημένων συνδέσεων, χρησιμοποιώντας το AES-128. Επειδή η διαδικασία της κρυπτογράφησης είναι χρονοβόρα και, κυρίως, καταναλώνει ενέργεια, στις περισσότερες BLE συσκευές, ειδικά αν πρόκειται να λειτουργούν με απλές μπαταρίες, η κρυπτογράφηση πραγματοποιείται από εξειδικευμένο hardware, το οποίο περιέχεται στο Link Layer. Στις περιπτώσεις, λοιπόν, εκείνες, στις οποίες ο host θέλει να κρυπτογραφήσει κάποια δεδομένα, αντί να το κάνει στον κεντρικό επεξεργαστή της συσκευής, ζητά από τον controller να τα κρυπτογραφήσει, με τη βοήθεια του εξειδικευμένου hardware που πιθανόν περιλαμβάνει, ώστε να μειώνεται η κατανάλωση ενέργειας.

2.5 Logical Link Control and Adaptation Protocol (L2CAP)

Το Logical Link Control and Adaptation Protocol (L2CAP) του BLE είναι το ίδιο πρωτόκολλο με αυτό που χρησιμοποιείται στο Bluetooth Classic, με τη διαφορά ότι, στο BLE, χρησιμοποιείται με πολύ απλούστερο τρόπο. Όπως έχουμε ήδη αναφέρει, ο ρόλος του L2CAP είναι η πολύπλεξη των δεδομένων των πρωτόκολλων των ανώτερων επίπεδων, καθώς και η διάσπαση και ανασύνθεση πακέτων δεδομένων, τα οποία είναι μεγαλύτερα από το μέγιστο μέγεθος πακέτου του Link Layer. Αυτό, στο BLE, είναι 27 bytes για πακέτα δεδομένων. Τα advertising πακέτα στέλνονται απευθείας από το Link Layer, οπότε δεν περνάνε από το L2CAP. Η λειτουργία του L2CAP βασίζεται στην έννοια του καναλιού. Κάθε κανάλι έχει ένα αναγνωριστικό (Channel Identifier – CID), το οποίο χρησιμοποιείται από το L2CAP για τη διάκριση των διαφορετικών καναλιών. Το CID είναι ένας αριθμός 16 bit. Δύο συσκευές μπορούν να έχουν πολλά κανάλια ενεργά ταυτόχρονα. Η πολύπλεξη των δεδομένων των πρωτοκόλλων επιτυγχάνεται με τη χρήση διαφορετικών καναλιών για το καθένα.

Όπως είδαμε, στο Bluetooth Classic, τα L2CAP κανάλια μπορούν να είναι connectionless ή connection-oriented και μπορούν, επίσης, να δημιουργούνται και να απελευθερώνονται ανάλογα με τις ανάγκες των ανώτερων επίπεδων. Στο BLE, αντίθετα, για λόγους απλότητας, χρησιμοποιούνται προκαθορισμένα κανάλια, με συγκεκριμένα CIDs, καθένα από τα οποία αναλαμβάνει τη μεταφορά των δεδομένων κάποιου από τα ανώτερα πρωτόκολλα της στοίβας.

CID	**Χρήση**
0x0000	Null identifier: Reserved
0x0001	Κανάλι σηματοδοσίας Bluetooth Classic L2CAP
0x0002	Connectionless κανάλι (Bluetooth Classic)
0x0003	AMP manager protocol (Bluetooth Classic)
0x0004	Attribute protocol
0x0005	Κανάλι σηματοδοσίας BLE L2CAP
0x0006	Security Manager protocol
0x0007 - 0x003E	Reserved
0x003F	AMP test protocol (Bluetooth Classic)
0x0040 – 0xFFFF	Connection-oriented κανάλια. Dynamically allocated.

Τα κανάλια που χρησιμοποιούνται από το BLE είναι το 4 για τα δεδομένα του Attribute protocol, το 6 για τα δεδομένα του Security Manager protocol και το 5 για την αποστολή εντολών που ορίζονται από το L2CAP. Με την έκδοση 4.1 του BLE, δίνεται πλέον η δυνατότητα, και σε BLE εφαρμογές, να δημιουργούν κανάλια δυναμικά, τα οποία μπορούν να χρησιμοποιηθούν για μεταφορά δεδομένων με μεγαλύτερο throughput, αποφεύγοντας τη χρήση του Attribute protocol. Τα επιπλέον κανάλια που είναι διαθέσιμα είναι τα 0x0040 ως 0x007F. Το νέο χαρακτηριστικό του BLE, το οποίο χρησιμοποιεί τα επιπλέον κανάλια, ονομάζεται LE Credit Based Flow Control Mode.

Η μορφή του πακέτου του L2CAP είναι η ακόλουθη:

Length 2 bytes	**CID** 2 bytes	**Data** 0 – 65535 bytes

Τα τέσσερα πρώτα bytes αποτελούν το βασικό L2CAP header, το οποίο υπάρχει σε όλα τα πακέτα L2CAP. Ορισμένοι τύποι πακέτων L2CAP χρησιμοποιούν επιπλέον bytes. Στο BLE, το συγκεκριμένο overhead των τεσσάρων bytes, σημαίνει ότι το ωφέλιμο φορτίο, που μπορεί να σταλεί σε ένα πακέτο του Link Layer, είναι 27 – 4 = 23 bytes.

Πριν την έκδοση 4.1, υπήρχε μόνο μία εντολή που μπορούσε να σταλεί μέσω του L2CAP, και συνολικά 3 είδη signaling πακέτων:

- Command Reject
- Connection Parameter Update Request
- Connection Parameter Update Response

Όπως είδαμε παραπάνω, σε κάθε σύνδεση, ο master είναι αυτός που καθορίζει τις παραμέτρους, χωρίς να υπάρχει δυνατότητα διαπραγμάτευσης από τον slave. Ο μόνος τρόπος με τον οποίο ο slave μπορεί να αλλάξει τις παραμέτρους της σύνδεσης, είναι να στείλει στον master τις επιθυμητές τιμές, και ο master, αν τις δεχτεί, να ενημερώσει τις παραμέτρους της σύνδεσης με τα νέα δεδομένα. Σε κάθε περίπτωση, ο master μπορεί να απορρίψει ή να αλλάξει τις επιθυμητές από τον slave παραμέτρους. Για την αποστολή των παραμέτρων αυτών από τον slave, χρησιμοποιείται η εντολή Connection Parameter Update Request του L2CAP, αφού, στην έκδοση 4.0, δεν παρέχεται άλλος τρόπος σε κατώτερο επίπεδο.

Το τελευταίο άλλαξε με την έκδοση 4.1, στην οποία δίνεται η δυνατότητα στον host του slave να αλλάξει τις παραμέτρους της σύνδεσης, χρησιμοποιώντας νέες εντολές του HCI και του Link Layer. Επίσης προστέθηκαν επιπλέον εντολές στο L2CAP για την υποστήριξη του LE Credit Based Flow Control Mode. Αυτές είναι οι ακόλουθες:

- Disconnection request
- Disconnection response
- LE Credit Based Connection request
- LE Credit Based Connection response
- LE Flow Control Credit

2.6 Security Manager Protocol (SMP)

Το Security Manager Protocol (SMP) ορίζει τις διαδικασίες, με τις οποίες οι BLE συσκευές μπορούν να δημιουργήσουν και να ανταλλάξουν κλειδιά κρυπτογράφησης. Τα κλειδιά αυτά μπορούν να χρησιμοποιηθούν για λειτουργίες, όπως κρυπτογράφηση των δεδομένων που ανταλλάσσονται σε μια σύνδεση, πιστοποίηση αυθεντικότητας του αποστολέα, ενώ δίνεται και η δυνατότητα απόκρυψης της δημόσιας διεύθυνσης μιας συσκευής, ώστε να αποτρέπεται η παρακολούθησή της από κακόβουλους χρήστες. Το τελευταίο σχετίζεται με ζητήματα ιδιωτικότητας των χρηστών. Αν, δηλαδή, ένας χρήστης έχει μια BLE συσκευή πάνω του και αυτή εκτελεί advertising, καθίσταται δυνατή η παρακολούθηση των κινήσεών του, μέσω της λήψης των μηνυμάτων advertising της συσκευής. Με την απόκρυψη της δημόσιας διεύθυνσής της, δεν είναι πλέον δυνατό να γίνει η αντιστοίχηση της συσκευής με τον συγκεκριμένο χρήστη.

Το SMP περιγράφει όλες τις διαδικασίες και τους αλγόριθμους που χρησιμοποιούνται για την επίτευξη του, απαιτούμενου από την εκάστοτε εφαρμογή, επίπεδου ασφάλειας. Ο αλγόριθμος κρυπτογράφησης που χρησιμοποιείται είναι ο AES-128. Επίσης, το SMP σχετίζεται άμεσα με λειτουργίες του προτύπου, που βασίζονται σε κλειδιά κρυπτογράφησης, όπως οι διαδικασίες pairing και bonding. Κατά τη διαδικασία του pairing, δύο συσκευές ανταλλάσσουν κατάλληλα κλειδιά, ώστε να μπορούν να επικοινωνούν μεταξύ τους πάνω από ένα ασφαλές, κρυπτογραφημένο κανάλι (link). Αν τα κλειδιά αυτά αποθηκευτούν στις συσκευές, τότε αυτές μπορούν να τα χρησιμοποιήσουν ξανά, κατά την επόμενη σύνδεση, αναγνωρίζοντας, με τη βοήθειά τους, την απομακρυσμένη συσκευή. Σε αυτήν την περίπτωση οι συσκευές θεωρούνται bonded.

Σε αυτό το σημείο, υπεισέρχεται και η έννοια του authenticated link, το οποίο καθορίζεται από τη δυνατότητα προστασίας από επιθέσεις τύπου Man In The Middle (MITM). Ένα authenticated link παρέχει προστασία από επιθέσεις MITM. Το αν ένα κανάλι θεωρείται authenticated ή όχι, εξαρτάται από τον τρόπο δημιουργίας του. Με βάση αυτό το διαχωρισμό των καναλιών, το GAP ορίζει πέντε επίπεδα ασφάλειας, χωρισμένα σε δύο κατηγορίες. Τα αναφέρουμε εδώ, μιας και σχετίζονται άμεσα με το SMP. Τα επίπεδα ασφάλειας είναι τα ακόλουθα:

- Security mode 1
    - Level 1: Καμία κρυπτογράφηση ή ασφάλεια. Είναι το αρχικό επίπεδο ασφάλειας για νέες συνδέσεις. Μετά τη σύνδεσή τους, οι συσκευές μπορούν να επιλέξουν το επιθυμητό επίπεδο ασφάλειας.
    - Level 2: Κρυπτογράφηση πάνω από unauthenticated link. Δεν παρέχει προστασία από επιθέσεις MITM.
    - Level 3: Κρυπτογράφηση πάνω από authenticated link. Παρέχει προστασία από επιθέσεις MITM. Είναι το ανώτερο επίπεδο ασφάλειας.

- Security mode 2
    - Level 1: Απαίτηση ψηφιακής υπογραφής των δεδομένων και αποστολής τους μέσω unauthenticated link. Δεν παρέχει προστασία από επιθέσεις MITM.
    - Level 2: Απαίτηση ψηφιακής υπογραφής των δεδομένων και αποστολής τους μέσω authenticated link. Παρέχει προστασία από επιθέσεις MITM.

Η διαδικασία του pairing εκτελείται με τρόπο που εξαρτάται από τις δυνατότητες I/O των συσκευών. Οι δυνατότητες I/O που ορίζονται στο πρότυπο είναι οι ακόλουθες:

- No I/O
- Display Only
- Display Yes/No (δυνατότητα output στην οθόνη και απλής επιλογής από τον χρήστη)
- Keyboard Only
- Keyboard Display

Από τον συνδυασμό των παραπάνω στις συσκευές, προκύπτει η μέθοδος pairing που εφαρμόζεται. Στο πρότυπο προβλέπονται τρεις μέθοδοι pairing:

- **Just Works**: Χρησιμοποιείται, όταν δεν είναι δυνατή η μεταφορά, από τον χρήστη, πληροφοριών μεταξύ των συσκευών, όπως για παράδειγμα στην περίπτωση που καμία από τις συσκευές δεν παρέχει δυνατότητες I/O. Το Temporary Key, που είναι απαραίτητο για την εκκίνηση της διαδικασίας, τίθεται στην τιμή μηδέν. Δημιουργείται unauthenticated link.

- **Passkey Entry**: Μια από τις δυο συσκευές παράγει το Temporary Key, το οποίο ο χρήστης μεταφέρει στη δεύτερη συσκευή. Μπορεί να χρησιμοποιηθεί σε περιπτώσεις που οι συσκευές παρέχουν κατάλληλες δυνατότητες I/O για τη μεταφορά του Temporary Key. Δημιουργείται authenticated link.

- **Out Of Band (OOB)**: Σε αυτήν την περίπτωση, τα δεδομένα που χρησιμοποιούνται για τη δημιουργία των κλειδιών που απαιτούνται, ορίζονται και ανταλλάσσονται με τρόπο εξωτερικό του BLE. Για παράδειγμα, θα μπορούσε το Temporary Key να μεταφερθεί μεταξύ συσκευών, με επικοινωνία μέσω NFC (Near field communication). Δημιουργείται authenticated link.

Οι συνδυασμοί των δυνατοτήτων I/O και η αντιστοίχησή τους σε μεθόδους pairing παρουσιάζεται στον παρακάτω πίνακα:

Δυνατότητες I/O	No I/O	Display Only	Display Yes/No	Keyboard Only	Keyboard Display
No I/O	Just Works	Just Works	Just Works	Just Works	Just Works
Display Only	Just Works	Just Works	Just Works	Passkey Entry	Passkey Entry
Display Yes/No	Just Works	Just Works	Just Works	Passkey Entry	Passkey Entry
Keyboard Only	Just Works	Passkey Entry	Passkey Entry	Passkey Entry	Passkey Entry
Keyboard Display	Just Works	Passkey Entry	Passkey Entry	Passkey Entry	Passkey Entry

Οι κυριότεροι τύποι κλειδιών που δημιουργούνται και ανταλλάσσονται από τις BLE συσκευές είναι οι ακόλουθοι:

- **Temporary Key (TK):** Χρησιμοποιείται, όπως είδαμε, κατά τη διαδικασία του pairing. Στις συνηθισμένες περιπτώσεις, γίνεται μεταφορά μεταξύ των συσκευών, με κατάλληλες ενέργειες του χρήστη, μιας αριθμητικής τιμής από 0 ως 999999. Η τιμή αυτή τίθεται σε μηδέν στην περίπτωση της μεθόδου “Just Works”. Το TK χρησιμοποιείται από τις συσκευές για τη δημιουργία και ανταλλαγή του Short Term Key.

- **Short Term Key (STK):** Χρησιμοποιείται κατά την πρώτη ενεργοποίηση της κρυπτογράφησης μεταξύ των συσκευών, προκειμένου να δημιουργηθεί ένα κρυπτογραφημένο κανάλι, πάνω στο οποίο μπορούν να σταλούν τα υπόλοιπα κλειδιά.

- **Long Term Key (LTK):** Δημιουργείται από τις συσκευές κατά τη διαδικασία του pairing και ανταλλάσσεται μέσω κρυπτογραφημένης σύνδεσης. Χρησιμοποιείται για τη δημιουργία των Session Keys, τα οποία, στη συνέχεια, χρησιμοποιούνται για την κρυπτογράφηση των δεδομένων σε κάθε σύνδεση. Στην περίπτωση των bonded συσκευών, το LTK αποθηκεύεται.

- **Identity Resolving Key (IRK):** Δημιουργείται από μια συσκευή, στην οποία έχει ενεργοποιηθεί η απόκρυψη της δημόσιας διεύθυνσης, και χρησιμοποιείται από άλλες συσκευές για την αναγνώριση της. Οι συσκευή αυτή χρησιμοποιεί μια διεύθυνση, η οποία αποτελείται από ένα τυχαίο αριθμό και το κρυπτογράφημα αυτού, μέσω του IRK (resolvable private address). Η διεύθυνση αυτή μεταβάλλεται συχνά. Η δεύτερη συσκευή ελέγχει τις διευθύνσεις των advertisers, που περιέχονται στα advertising πακέτα, χρησιμοποιώντας το IRK. Αν ο έλεγχος επιτύχει, θεωρείται ότι βρέθηκε η συγκεκριμένη συσκευή, στην οποία αντιστοιχεί το IRK.

- **Connection Signature Resolving Key (CSRK):** Χρησιμοποιείται στις περιπτώσεις στις οποίες απαιτείται πιστοποίηση αυθεντικότητας του αποστολέα ενός πακέτου. Αυτή πραγματοποιείται με ψηφιακή υπογραφή των δεδομένων με χρήση του CSRK.

2.7 Generic Access Profile (GAP)

Το Generic Access Profile (GAP) ορίζει ρόλους, τρόπους λειτουργίας (modes) και διαδικασίες (procedures), μέσω των οποίων δύο, συμβατές με το πρότυπο, συσκευές μπορούν να ανακαλύψουν η μία την άλλη, να συνδεθούν και να ανταλλάξουν δεδομένα. Το GAP είναι από τα σημαντικότερα στοιχεία της στοίβας πρωτοκόλλων του Bluetooth, και συνδέεται με τα υπόλοιπα με ποικίλους τρόπους, όπως είδαμε και παραπάνω. Το GAP είναι κοινό μεταξύ των BLE και Bluetooth Classic, αλλά οι λειτουργίες του σε καθένα από αυτά είναι σε μεγάλο βαθμό διαφορετικές. Εμείς θα ασχοληθούμε μόνο με το τμήμα του GAP που αφορά το BLE.

2.7.1 Ρόλοι

Στο GAP ορίζονται τέσσερις ρόλοι με τους οποίους μπορεί να εμφανίζεται μια BLE συσκευή. Μια συσκευή έχει τη δυνατότητα να βρίσκεται σε περισσότερους από ένα ρόλους ταυτόχρονα, αν κάτι τέτοιο υποστηρίζεται από τον controller της. Οι ρόλοι των συσκευών στο GAP είναι οι εξής:

- **Broadcaster**: Ο broadcaster είναι μια συσκευή, η οποία στέλνει πακέτα advertising. Αυτός ο ρόλος χρησιμοποιείται συνήθως από συσκευές που μεταδίδουν δεδομένα περιοδικά, για ενδιαφερόμενες συσκευές, οι οποίες βρίσκονται, ενδεχομένως, στην εμβέλειά τους, και έχουν το ρόλο του observer. Η συσκευή δεν απαιτείται να έχει κύκλωμα λήψης, αλλά μόνο αποστολής.

- **Observer**: Ο observer είναι μια συσκευή, η οποία ελέγχει τα advertising κανάλια για ενδεχόμενα πακέτα advertising από broadcasters, με δεδομένα που μπορεί να την ενδιαφέρουν (Link Layer scanner). Η συσκευή δεν απαιτείται να έχει κύκλωμα αποστολής, αλλά μόνο λήψης.

- **Peripheral**: Ένα peripheral είναι μια συσκευή, η οποία εκτελεί τη διαδικασία του advertising, στέλνοντας πακέτα που δηλώνουν ότι είναι διαθέσιμη προς σύνδεση με άλλες συσκευές. Μετά από τη δημιουργία της σύνδεσης, το peripheral μετέχει σε αυτή με το ρόλο του Link Layer slave. Η συσκευή απαιτείται να έχει κυκλώματα και αποστολής και λήψης.

- **Central**: Ένα central είναι μια συσκευή, η οποία ελέγχει τα advertising κανάλια για ενδεχόμενα πακέτα advertising από peripherals, προκειμένου να συνδεθεί με αυτά (Link Layer initiator). Μετά από τη δημιουργία της σύνδεσης, το central μετέχει σε αυτή με το ρόλο του Link Layer master. Η συσκευή απαιτείται να έχει κυκλώματα και αποστολής και λήψης.

2.7.2 Τρόποι λειτουργίας (modes) και διαδικασίες (procedures)

Στο GAP ορίζονται τρόποι λειτουργίας (modes), στους οποίους μπορεί να βρίσκεται μια συσκευή, και αντίστοιχες διαδικασίες (procedures), που μπορούν να εκτελούνται σε κάθε mode. Οι τρόποι λειτουργίας και οι αντίστοιχες διαδικασίες χωρίζονται σε τέσσερις κατηγορίες.

- Broadcast modes and procedures
- Discovery modes and procedures
- Connection modes and procedures
- Bonding modes and procedures

2.7.2.1 Broadcast modes and procedures

Υπάρχει μόνο ένα broadcast mode, με αυτό το όνομα, και ένα procedure, που ονομάζεται observation procedure. Μια συσκευή σε broadcast mode στέλνει δεδομένα, χωρίς να απαιτείται σύνδεση, χρησιμοποιώντας τα advertising κανάλια του BLE. Τα δεδομένα πρέπει να έχουν τη μορφή που ορίζεται στο GAP, την οποία περιγράψαμε παραπάνω. Το observation procedure εκτελείται από μία συσκευή, προκειμένου να λάβει αυτά τα δεδομένα.

2.7.2.2 Discovery modes and procedures

Υπάρχουν τρία discovery modes και τρία discovery procedures. Τα discovery modes είναι τα ακόλουθα:

- **Non-Discoverable mode**: Μια συσκευή σε non-discoverable mode δεν μπορεί να ανακαλυφθεί από άλλες συσκευές που εκτελούν τις διαδικασίες general discovery procedure ή limited discovery procedure, οι οποίες περιγράφονται παρακάτω. Η επιλογή του mode γίνεται με βάση τις σημαίες, που περιέχονται στο "Flags" AD type των πακέτων advertising. Προκειμένου μια συσκευή να είναι discoverable, πρέπει να θέσει την αντίστοιχη σημαία στα πακέτα advertising που στέλνει, ενεργοποιώντας ένα από τα επόμενα modes.
- **Limited Discoverable mode**: Μια συσκευή σε limited discoverable mode μπορεί να ανακαλυφθεί από άλλες συσκευές, για κάποιο σύντομο χρονικό διάστημα, με εκτέλεση από αυτές των general ή limited discovery procedures. Το μέγιστο χρονικό διάστημα παραμονής σε αυτό το mode ήταν 30 sec στην έκδοση 4.0, ενώ αυξήθηκε σε 180 sec στην 4.1. Στη συνηθισμένη περίπτωση, μια συσκευή τίθεται σε limited discoverable mode μετά από κάποια ενέργεια του χρήστη, προκειμένου αυτή να βρεθεί από κάποια άλλη συσκευή του. Για αυτόν τον λόγο, στα advertising πακέτα των συσκευών σε αυτό το mode, περιλαμβάνονται, συνήθως, πληροφορίες όπως το όνομα της συσκευής ή η ισχύς εκπομπής της. Αυτά μπορούν να χρησιμοποιηθούν από το user interface της δεύτερης συσκευής, προκειμένου να ταξινομήσει τις συσκευές που ανακαλύπτει, ώστε η συσκευή σε limited discoverable mode, που πιθανώς είναι αυτή με την οποία ενδιαφέρεται να συνδεθεί ο χρήστης, να εμφανίζεται ψηλά στη συγκεκριμένη λίστα. Το advertising interval τίθεται, συνήθως, σε σχετικά χαμηλές τιμές.
- **General Discoverable mode**: Είναι το discoverable mode που χρησιμοποιείται στη γενική περίπτωση, κατά την οποία μια συσκευή θέλει να δηλώσει την παρουσία της σε ενδιαφερόμενες συσκευές, μέσω αποστολής πακέτων advertising. Το advertising interval τίθεται, συνήθως, σε σχετικά μεγάλες τιμές (> 1 sec).

Τα τρία discovery procedures που προβλέπονται από το πρότυπο είναι τα ακόλουθα:

- **General Discovery Procedure**: Μια συσκευή εκτελεί το general discovery procedure, προκειμένου να ανακαλύψει συσκευές που βρίσκονται σε general ή limited discoverable mode. Αυτό γίνεται εκτελώντας τη λειτουργία scanning στο Link Layer, με τον τρόπο που περιγράφεται παραπάνω.
- **Limited Discovery Procedure**: Μια συσκευή εκτελεί το limited discovery procedure, προκειμένου να ανακαλύψει μόνο συσκευές που βρίσκονται σε limited discoverable mode. Η διαδικασία είναι ίδια με αυτή του general discovery procedure, με τη διαφορά ότι γίνεται τοπικά έλεγχος των "Flags" στα πακέτα advertising, ώστε μόνο συσκευές σε limited discoverable mode να παρουσιάζονται στον χρήστη.
- **Name Discovery Procedure**: Μια συσκευή εκτελεί το name discovery procedure, προκειμένου να ανακτήσει το GAP Device Name μιας απομακρυσμένης συσκευής.

Για την πραγματοποίηση της διαδικασίας, απαιτείται να γίνει σύνδεση με την απομακρυσμένη συσκευή. Το GAP Device Name περιέχεται, όπως θα δούμε στο επόμενο κεφάλαιο, σε ένα, μοναδικό στη συσκευή, GATT characteristic, και μπορεί να διαβαστεί άμεσα.

2.7.2.3 Connection modes and procedures

Υπάρχουν τρία connection modes και έξι connection procedures. Τα connection modes είναι τα ακόλουθα:

- **Non-Connectable Mode**: Μια συσκευή σε non-connectable mode δεν επιτρέπει συνδέσεις με άλλες συσκευές. Ένας Broadcaster ή Observer είναι πάντοτε σε non-connectable mode. Σε αυτό το mode, ένας advertiser, μπορεί να εκτελεί μόνο non-connectable advertising events (ADV_NONCONN_IND, ADV_SCAN_IND).
- **Directed Connectable Mode**: Μια συσκευή σε directed-connectable mode είναι ένα peripheral, το οποίο, ως advertiser, εκτελεί directed connectable advertising events (ADV_DIRECT_IND), προκειμένου να συνδεθεί άμεσα με μια γνωστή συσκευή, η οποία εκτελεί το auto ή το general connection establishment procedure. Δεν επιτρέπονται συνδέσεις με άλλες συσκευές.
- **Undirected Connectable Mode**: Μια συσκευή σε undirected connectable mode είναι ένα peripheral, το οποίο επιτρέπει τη σύνδεση με άλλες συσκευές, που εκτελούν το auto ή το general connection establishment procedure. Είναι το connection mode που χρησιμοποιείται στη γενική περίπτωση, από συσκευές peripheral που ενδιαφέρονται να συνδεθούν με άλλες συσκευές. Μια συσκευή σε undirected connectable mode εκτελεί, ως advertiser, connectable undirected advertising events (ADV_IND).

Τα έξι connection procedures που προβλέπονται από το πρότυπο είναι τα ακόλουθα:

- **Auto Connection Eestablishment Procedure**: Επιτρέπει σε μια συσκευή central να συνδέεται αυτόματα με ένα σύνολο από γνωστές συσκευές, αν κάποια από αυτές είναι σε εμβέλεια και εκτελεί connectable advertising. Οι συσκευές ορίζονται με τη βοήθεια της white list που περιέχεται στο Link Layer. Οι παράμετροι σύνδεσης είναι κοινές για όλες τις συνδέσεις. Ο host δεν ενημερώνεται πριν την έναρξη της διαδικασίας σύνδεσης, παρά μόνο αφού αυτή έχει ολοκληρωθεί. Το procedure αυτό χρησιμοποιείται συνήθως για την αυτόματη σύνδεση με ένα σύνολο παρόμοιων συσκευών. Δεν μπορεί να χρησιμοποιηθεί για απομακρυσμένες συσκευές με ενεργοποιημένη την απόκρυψη της διεύθυνσής τους, αφού, σε αυτήν την περίπτωση, δεν είναι δυνατόν να αναγνωριστούν μέσω της white list.
- **Selective Connection Establishment Procedure**: Επιτρέπει σε μια συσκευή central να ορίσει συγκεκριμένες συσκευές με τις οποίες ενδιαφέρεται να συνδεθεί. Όπως και προηγουμένως, η επιλογή αυτή γίνεται μέσω της white list. Ο controller ενημερώνει τον host μόνο για advertising events, που εκτελούνται από συσκευές που περιέχονται στη white list. Οι υπόλοιπες συσκευές, που ενδεχομένως κάνουν advertising, αγνοούνται. Ο host, αν ενδιαφέρεται να συνδεθεί με μια συσκευή για την οποία ενημερώθηκε από τον controller, εκτελεί, στη συνέχεια, το direct connection establishment procedure. Όπως και στην προηγούμενη περίπτωση, δεν μπορεί να χρησιμοποιηθεί για απομακρυσμένες συσκευές με ενεργοποιημένη την απόκρυψη της διεύθυνσής τους.
- **General Connection Establishment Procedure**: Είναι η γενική διαδικασία με την οποία μια συσκευή central μπορεί να συνδεθεί με άλλες συσκευές, που βρίσκονται σε directed ή undirected connectable mode. Δε χρησιμοποιείται η white list και γίνεται έλεγχος όλων των πακέτων advertising που λαμβάνονται. Ταυτόχρονα, είναι δυνατόν

να γίνεται έλεγχος των διευθύνσεων, μέσω IRKs γνωστών συσκευών, που είναι αποθηκευμένα στο central. Με αυτόν τον τρόπο μπορούν να αναγνωριστούν συσκευές με ενεργοποιημένη την απόκρυψη της διεύθυνσής τους. Αν ο host ενδιαφέρεται να συνδεθεί με μια συσκευή που προέκυψε από αυτή τη διαδικασία, εκτελεί, στη συνέχεια, το direct connection establishment procedure.

- **Direct Connection Establishment Procedure**: Μια συσκευή εκτελεί το direct connection establishment procedure, προκειμένου να συνδεθεί με μια συγκεκριμένη απομακρυσμένη συσκευή. Η συσκευή είναι central στο GAP και initiator στο Link Layer, ενώ η απομακρυσμένη συσκευή είναι ένα peripheral, που έχει ανακαλυφθεί νωρίτερα με διαδικασία αναζήτησης. Το central θέτει τις επιθυμητές παραμέτρους και εκκινεί τη διαδικασία της σύνδεσης, η οποία οδηγεί στην αποστολή ενός πακέτου CONNECT_REQ από το Link Layer. Όπως είδαμε παραπάνω, τα selective και general connection establishment procedure χρησιμοποιούν αυτό το procedure για τη δημιουργία της σύνδεσης.
- **Connection Parameter Update Procedure**: Μια συσκευή εκτελεί το connection parameter update procedure, προκειμένου να αλλάξει τις παραμέτρους της σύνδεσής της με μια απομακρυσμένη συσκευή. Οι παράμετροι είναι αυτές που είδαμε παραπάνω, κατά την περιγραφή του Link Layer. Η συγκεκριμένη διαδικασία μπορεί να εκτελεστεί τόσο από ένα peripheral, το οποίο είναι ο slave της σύνδεσης, όσο και από ένα central, το οποίο είναι ο master της σύνδεσης, με χρήση κατάλληλων εντολών HCI και L2CAP που παρέχονται στους hosts, και οι οποίες μεταφράζονται σε μηνύματα που ανταλλάσσονται μεταξύ των Link Layers. Σε κάθε περίπτωση, η τελική απόφαση για την αλλαγή των παραμέτρων και τις τιμές τους ανήκει στο central (master).
- **Terminate Connection Procedure**: Μια συσκευή εκτελεί το terminate connection procedure, προκειμένου να τερματίσει τη σύνδεση με μια απομακρυσμένη συσκευή. Ο τερματισμός πραγματοποιείται μέσω της διαδικασίας που υπάρχει στο Link Layer για τον σκοπό αυτό.

2.7.2.4 Bonding modes and procedures

Υπάρχουν δύο bonding modes και ένα bonding procedure. Όπως αναφέραμε παραπάνω, το bonding μεταξύ συσκευών πραγματοποιείται με την ανταλλαγή και αποθήκευση κλειδιών κρυπτογράφησης. Η ανταλλαγή των κλειδιών κρυπτογράφησης γίνεται κατά το pairing των συσκευών. Τα bonding modes που ορίζονται στο πρότυπο είναι τα bondable και non-bondable mode. Μια συσκευή σε non-bondable mode δεν επιτρέπει σε άλλες συσκευές να γίνουν bonded με αυτή. Αντίθετα, μια συσκευή σε bondable mode επιτρέπει σε άλλες συσκευές να γίνουν bonded με αυτή, παρέχοντας τα κλειδιά που είναι απαραίτητα, και αποθηκεύοντας τα κλειδιά που παρέχονται από την απομακρυσμένη συσκευή. Το bonding procedure εκτελείται από συσκευές που θέλουν να γίνουν bonded με άλλες συσκευές, οι οποίες πρέπει να βρίσκονται σε bondable mode. Μπορεί να εκτελεστεί και κατά την προσπάθεια μιας απομακρυσμένης συσκευής να πραγματοποιήσει κάποια ενέργεια, η οποία απαιτεί bonding.

Κεφάλαιο 3

Bluetooth Low Energy
ATT, GATT, Profiles

Το Attribute Protocol (ATT) και το Generic Attribute Profile (GATT) είναι τα σημαντικότερα τμήματα της στοίβας πρωτοκόλλων του Bluetooth Low Energy, σε ό,τι αφορά τους developers BLE εφαρμογών. Είναι τα τμήματα εκείνα, πάνω στα οποία είναι χτισμένα τα APIs στις περισσότερες πλατφόρμες που υποστηρίζουν το BLE, και, ως εκ τούτου, τα τμήματα με τα οποία έρχονται σε άμεση επαφή οι developers. Όπως θα δούμε σε επόμενο κεφάλαιο, όταν θα περιγράψουμε το BLE API του Android, πολλές από τις λειτουργίες που παρέχονται από αυτό, αντιστοιχούν επακριβώς σε λειτουργίες του GATT.

Τα ATT και GATT αποτελούν το μοντέλο δεδομένων του BLE, πάνω στο οποίο μπορούν να δημιουργηθούν οι διάφορες εφαρμογές και περιπτώσεις χρήσης. Το ATT ορίζει τις βασικές έννοιες και τα δομικά στοιχεία του μοντέλου δεδομένων. Πάνω σε αυτά το GATT χτίζει μια ιεραρχία από δομές δεδομένων, δίνοντας δομή και σημασιολογία στην επίπεδη οργάνωση του ATT. Με τη βοήθεια αυτών, μπορούν, στη συνέχεια, να οριστούν τα λεγόμενα GATT-based profiles. Αυτά μπορούν να θεωρηθούν ως ο τυπικός ορισμός του επιπέδου εφαρμογής για μια συγκεκριμένη περίπτωση χρήσης.

3.1 Attribute Protocol

Το Attribute Protocol (ATT) ορίζει μια βασική δομή, το attribute, και τις διαδικασίες με τις οποίες μία συσκευή μπορεί να ανακαλύψει τα attributes, που βρίσκονται αποθηκευμένα σε μια απομακρυσμένη συσκευή, και, στη συνέχεια, να τα διαβάσει, να τα μεταβάλει ή να δεχτεί ειδοποιήσεις σχετικά με αυτά. Οι δύο αυτές συσκευές αποτελούν και τους δύο ρόλους που ορίζονται στο πρωτόκολλο, δηλαδή τους server και client. Ο ATT server είναι η συσκευή που περιέχει τα attributes, ενώ ο ATT client είναι η συσκευή που επικοινωνεί με το server, χρησιμοποιώντας το Attribute Protocol, προκειμένου να εκτελέσει τις παραπάνω λειτουργίες.

3.1.1 Attribute

Ένα attribute είναι μία διακριτή τιμή με τρεις ιδιότητες:

- Attribute type
- Attribute handle
- Permissions

Handle (16 bit)	**Type (UUID)** (16bit-32bit-128bit)	**Value** (0-512 byte)	**Permissions**

Η παραπάνω αναπαράσταση είναι ενδεικτική. Το πρότυπο δεν ορίζει τον τρόπο αποθήκευσης των attributes. Σε νοερό επίπεδο, όμως, μπορούμε να θεωρήσουμε ότι τα attributes αποθηκεύονται σε μία επίπεδη βάση δεδομένων (flat database) με ένα μόνο table.

3.1.1.1 Attribute Value

Η τιμή ενός attribute είναι ένας πίνακας από bytes, που μπορεί να έχει σταθερό ή μεταβλητό μέγεθος, με μέγιστο τα 512 bytes. Τα δεδομένα που αποθηκεύονται στην τιμή του attribute είναι αδιάφορα για το πρωτόκολλο, και ορίζονται από ανώτερα επίπεδα, συνήθως σε συνάρτηση με το UUID, που δίνει τον τύπο του attribute. Ένα attribute value που δε χωράει σε ένα πακέτο (Protocol Data Unit – PDU), μπορεί να σταλεί σε περισσότερα πακέτα.

Αξίζει να σημειωθεί ότι, για λόγους οικονομίας, κατά την αποστολή τιμών attributes, το μέγεθος της τιμής δεν περιέχεται στα δεδομένα που αποστέλλονται. Οπότε, αν ένα attribute έχει τιμή μεταβλητού μεγέθους, άρα ο client δε γνωρίζει εξαρχής το μέγεθός του, το τελευταίο θα πρέπει να μπορεί να προκύψει από το μήκος των δεδομένων στο πακέτο που ελήφθη. Αυτό δημιουργεί κάποιους περιορισμούς στην αποστολή τιμών attributes. Για παράδειγμα, αν στέλνονται πολλά attribute values με το ίδιο πακέτο, ένα value μεταβλητού μεγέθους πρέπει να είναι το μοναδικό στο πακέτο με μη σταθερό μέγεθος, και, επίσης, πρέπει να τοποθετείται στο τέλος του πακέτου. Γενικότερα, αν ο client δε γνωρίζει το μέγεθος ενός attribute value, αυτό, ακόμα και αν έχει σταθερό μέγεθος, πρέπει αναγκαστικά να αποστέλλεται σε ξεχωριστό πακέτο.

3.1.1.2 Attribute Handle

Το attribute handle είναι μία τιμή 16 bit, η οποία χρησιμεύει ως αναγνωριστικό για το συγκεκριμένο attribute. Κάθε attribute έχει το δικό του μοναδικό handle. Η τιμή μηδέν είναι δεσμευμένη και δεν επιτρέπεται, οπότε ο μέγιστος αριθμός attributes σε ένα ATT server είναι 65535. Στην πράξη, βέβαια, οι περισσότεροι ATT servers περιέχουν μερικές δεκάδες attributes. Οι τιμές των attributes δεν είναι απαραίτητο να είναι συνεχόμενες. Επίσης, δεν έχουν κάποιο νόημα ή σημασία, πέρα από τα πλαίσια της επικοινωνίας μεταξύ συγκεκριμένων client και server.

Τα attribute handles χρησιμοποιούνται σε όλες τις λειτουργίες του πρωτοκόλλου, όπου απαιτείται η επιλογή κάποιου attribute. Κατά μία έννοια λειτουργούν ως διευθύνσεις των attributes. Το μικρό μέγεθος των handles βοηθάει στη μείωση του κόστους επικοινωνίας.

Με τη βοήθεια των handles, δημιουργείται η μόνη ουσιαστική δομή πάνω στην, κατά τα άλλα, επίπεδη οργάνωση του ATT. Αυτή η δομή ονομάζεται Attribute Handle Grouping. Σύμφωνα με αυτή, μπορούν να οριστούν τύποι attributes, οι οποίοι θεωρούνται τύποι ομαδοποίησης (group types). Ένα group type attribute εκκινεί μία ομάδα από attributes, η οποία περιέχει όλα τα attributes, των οποίων τα handles βρίσκονται μεταξύ αυτών, του group type attribute που ξεκίνησε τη δική τους ομάδα, και του group type attribute που εκκινεί την επόμενη. Η τιμή δηλαδή των handles μπορεί να χρησιμοποιηθεί για να ορίσει handle ranges. Τα group type attributes ορίζονται από ανώτερα επίπεδα. Στην περιγραφή του GATT, θα δούμε πώς αυτό το χαρακτηριστικό του ATT χρησιμοποιείται για την υλοποίηση σύνθετων δομών δεδομένων.

Ένα άλλο χαρακτηριστικό του πρωτοκόλλου, που σχετίζεται με τα attribute handles, είναι το handle caching. Κατά τη διάρκεια της σύνδεσης, τα handles διατηρούν τις τιμές τους. Το handle caching δίνει τη δυνατότητα στους clients να αποθηκεύουν τοπικά τις τιμές των attribute handles, τις οποίες έχουν ανακαλύψει. Έτσι, αποφεύγεται η εκ νέου ανακάλυψή τους στην επόμενη σύνδεση, μια διαδικασία που απαιτεί την αποστολή πολλών πακέτων. Παρακάτω θα δούμε ορισμένους από τους κανόνες που ισχύουν στο handle caching.

3.1.1.3 Attribute Type

Ο τύπος ενός attribute καθορίζεται με τη βοήθεια ενός Universally Unique Identifier (UUID). Τα UUID είναι αναγνωριστικοί αριθμοί των 128 bit (16 bytes), οι οποίοι παράγονται με τέτοιο τρόπο, ώστε είναι δύσκολο να δημιουργηθούν όμοια UUIDs, για αυτό και θεωρούνται μοναδικοί. Χρησιμοποιούνται σε πολλά πρωτόκολλα και περιπτώσεις χρήσης. Συνήθως αναπαρίστανται, με τη λεγόμενη κανονική μορφή, ως 32 δεκαεξαδικά ψηφία, χωρισμένα με παύλες σε 5 ομάδες (8-4-4-4-12). Για παράδειγμα:

11223344-5566-7788-99aa-bbccddeeff00

Το Bluetooth SIG ορίζει μια σειρά από UUIDs για διάφορους τύπους attributes που υπάρχουν στο BLE. Ένας developer μπορεί να ορίσει επιπλέον UUIDs για τους δικούς του τύπους δεδομένων. Λόγω του μεγάλου μεγέθους των UUIDs, το πρότυπο δίνει τη δυνατότητα χρήσης μικρότερων μεγεθών, 16 bit (έκδοση 4.0) και 32 bit (έκδοση 4.1). Τέτοια είναι όλα τα UUIDs, που ορίζονται από το SIG, και χρησιμοποιούνται στο ίδιο το πρότυπο και στα διάφορα profiles που έχουν γίνει αποδεκτά ως πρότυπα (adopted profiles). Η μείωση αυτή του μεγέθους είναι σημαντική, καθώς μειώνει αντίστοιχα το χώρο που χρειάζεται για να αποθηκευτούν τα attributes, μειώνει το χρόνο αναζήτησης σε αυτά, και, το κυριότερο, μειώνει το κόστος επικοινωνίας, αφού δε χρειάζεται να στέλνονται ολόκληρα τα UUIDs.

Τα UUIDs του SIG ορίζονται ως τιμές 16 και 32 bit που αντιπροσωπεύουν τη διαφορά του εκάστοτε UUID, από μία κοινή βάση, το UUID:

00000000-0000-1000-8000-00805F9B34FB

Η διαφορά προστίθεται στο υπογραμμισμένο κομμάτι. Για παράδειγμα το 16-bit UUID του Alert Level characteristic value attribute είναι 0x2A06, το οποίο αντιστοιχεί στο 128-bit UUID:

00002A06-0000-1000-8000-00805F9B34FB

Μόνο τα profiles του SIG μπορούν να χρησιμοποιούν τα μικρά UUIDs. Τα custom profiles, εκτός και αν γίνουν κάποια στιγμή adopted, χρησιμοποιούν πλήρη UUIDs.

3.1.1.4 Permissions

Τα permissions ενός attribute ορίζουν τις επιτρεπτές ενέργειες που μπορούν να γίνουν πάνω σε αυτό. Τα attribute permissions είναι ένας συνδυασμός από permissions πρόσβασης (access), κρυπτογράφησης (encryption), πιστοποίησης αυθεντικότητας (authentication) και εξουσιοδότησης (authorization).

Τα access permissions αφορούν τη δυνατότητα του client να διαβάζει ή να γράφει στο συγκεκριμένο attribute, όπως επίσης το αν μπορεί να γίνει notification ή indication του attribute. Έτσι, μπορούμε να έχουμε permissions του τύπου:

- No access allowed
- Readable
- Writeable
- Readable and writable

Τα encryption permissions αφορούν το αν τα δεδομένα του attribute μπορούν να σταλούν πάνω από μη κρυπτογραφημένο κανάλι. Οι δυνατοί τύποι permissions είναι:

- No encryption required
- Encryption required

Αυτοί μπορούν να συνδυαστούν με τα authentication permissions, τα οποία καθορίζουν αν τα δεδομένα του attribute μπορούν να σταλούν πάνω από ένα κρυπτογραφημένο κανάλι, το οποίο δεν είναι authenticated, άρα δεν παρέχει προστασία από επιθέσεις τύπου MITM. Οι δυνατοί τύποι authentication permissions είναι:

- No Authentication Required
- Authentication Required

Τέλος, τα authorization permissions καθορίζουν αν η επιθυμητή ενέργεια πάνω στο attribute απαιτεί ή όχι authorization. Το BLE πρότυπο δεν ορίζει το ίδιο τι ακριβώς σημαίνει η έννοια της εξουσιοδότησης και πώς αυτή επιτυγχάνεται. Αυτά καθορίζονται σε ανώτερο επίπεδο, από την εκάστοτε εφαρμογή. Για παράδειγμα, μια εφαρμογή, πριν εκτελέσει κάποια ενέργεια, μπορεί να ζητά την άδεια ή κάποιον κωδικό από τον χρήστη. Οι δυνατοί τύποι permissions είναι:

- No Authorization Required
- Authorization Required

Όπως αναφέραμε παραπάνω, τα διάφορα permissions μπορούν να συνδυαστούν, για τη δημιουργία πιο σύνθετων permissions. Για παράδειγμα, ένα attribute μπορεί να readable μόνο πάνω από authenticated (άρα και encrypted) κανάλι, και writeable, επίσης πάνω από authenticated κανάλι, αλλά με επιπλέον ανάγκη authorization.

Τα attribute permissions ορίζονται σε ανώτερο επίπεδο και, παρότι αποθηκεύονται μαζί με τα attributes, δεν υπάρχει τρόπος να διαβαστούν μέσω του Attribute protocol. Ο μόνος τρόπος, με τον οποίο η τιμή κάποιων permissions μπορεί να γίνει ορατή στο επίπεδο του ATT, είναι με την προσπάθεια εκτέλεσης μιας μη επιτρεπτής διαδικασίας, οπότε θα ληφθεί το αντίστοιχο μήνυμα λάθους.

Είναι δυνατόν να οριστούν attributes, τα οποία να μην είναι readable, αλλά να είναι writeable. Τέτοιου τύπου attributes, τα οποία μπορούν επίσης να γίνουν και notified ή indicated, ονομάζονται control-point attributes. Η χρησιμότητά τους είναι μεγάλη σε διάφορες περιπτώσεις χρήσης. Για παράδειγμα, η εγγραφή μιας τιμής σε ένα control-point attribute μπορεί να θεωρηθεί ως εντολή για την εκκίνηση μιας λειτουργίας στον GATT server, της οποίας το αποτέλεσμα επιστρέφεται με indication του ίδιου attribute. Αυτή η χρήση εμφανίζεται συχνά στα adopted profiles και χρησιμοποιείται και από το δικό μας σύστημα.

3.1.2 Protocol Data Units

3.1.2.1 Δομή Protocol Data Unit

Προκειμένου να εκτελεστούν οι λειτουργίες που προβλέπονται από το πρωτόκολλο, γίνεται ανταλλαγή πακέτων (Protocol Data Unit – PDU) μεταξύ των client και server. Η PDU ξεκινάει με ένα byte που καθορίζει το είδος της, και τη λειτουργία για την οποία προορίζεται (opcode). Μετά το opcode, ακολουθούν οι παράμετροι της συγκεκριμένης λειτουργίας. Η PDU μπορεί επίσης να περιέχει στο τέλος της ένα authentication signature των 12 bytes, το οποίο χρησιμοποιείται για την πιστοποίηση αυθεντικότητας του opcode και των παραμέτρων.

Το μέγιστο μήκος δεδομένων της PDU, το οποίο ονομάζεται MTU (Maximum Transmission Unit), μπορεί να είναι αποτέλεσμα διαπραγμάτευσης μεταξύ client και server, με ελάχιστη δυνατή (και default) τιμή το 23. Αυτή η τιμή είναι ακριβώς το μέγιστο μέγεθος δεδομένων του ATT, που μπορούν να σταλούν σε ένα πακέτο του Link Layer.

31 bytes (LL data) - 4 bytes (MIC) - 4 bytes (L2CAP) = 23 bytes

Αφαιρώντας και το αρχικό byte της PDU, παίρνουμε ένα μέγιστο ωφέλιμο φορτίο για το ATT στα 22 bytes, αν χρησιμοποιείται η default MTU. Αυτό μειώνεται στα 10 bytes στην περίπτωση που υπάρχει και authentication signature.

3.1.2.2 Τύποι PDUs

Οι PDUs και οι αντίστοιχες λειτουργίες του ATT χωρίζονται σε 6 τύπους:

- **Requests**: PDUs που στέλνονται από τον client στον server και προκαλούν την αποστολή responses.
- **Responses**: PDUs που στέλνονται από τον server στον client ως απαντήσεις σε προηγούμενα requests.
- **Commands**: PDUs που στέλνονται από τον client στον server, χωρίς ο δεύτερος να στέλνει απάντηση.
- **Notifications**: PDUs που στέλνονται από τον server στον client, χωρίς να επιβεβαιώνεται η λήψη τους.
- **Indications**: PDUs που στέλνονται από τον server στον client, με επιβεβαίωση της λήψης τους από τον δεύτερο, με αποστολή confirmations.
- **Confirmations**: PDUs που στέλνονται από τον client στον server για επιβεβαίωση λήψης προηγούμενων indications.

Ένα ζεύγος request-response ή ένα ζεύγος indication-confirmation ορίζουν ένα transaction. Μόνο ένα transaction μπορεί να είναι κάθε στιγμή ενεργό σε κάθε πλευρά. Δηλαδή, ο client, αν έχει ήδη στείλει ένα request, δεν μπορεί να στείλει δεύτερο, αν πρώτα δε λάβει το response από τον server. Αντίστοιχα, αν ο server έχει ήδη στείλει ένα indication, δεν μπορεί να στείλει δεύτερο, αν δε λάβει πρώτα το confirmation του client. Ο περιορισμός ισχύει για κάθε πλευρά ξεχωριστά. Για παράδειγμα, ο server μπορεί να στείλει indication σε έναν client ο οποίος περιμένει από αυτόν ένα response. Το πρότυπο ορίζει ένα χρονικό όριο 30 δευτερολέπτων για την ολοκλήρωση ενός transaction.

Τα command και notification, δεν ορίζουν transaction και μπορούν να στέλνονται οποιαδήποτε στιγμή, ακόμα και αν η αντίστοιχη πλευρά έχει ήδη κάποιο ενεργό transaction. Η έλλειψη απάντησης από την άλλη πλευρά σημαίνει ότι τα πακέτα αυτά δεν είναι σίγουρο ότι έχουν όντως ληφθεί. Εδώ, βέβαια, πρέπει να πούμε ότι όλα τα πακέτα που στέλνονται από το Link Layer επιβεβαιώνονται. Δηλαδή, η απώλεια ενός command ή notification δε συμβαίνει στη μεταφορά, αλλά μετά τη λήψη τους, αν, για παράδειγμα, ο host του peer έχει μεγάλο αριθμό pending πακέτων και η αντίστοιχη ουρά είναι γεμάτη.

3.1.3 Λειτουργίες Attribute Protocol

Σε αυτή την υπο-ενότητα περιγράφουμε συνοπτικά τις λειτουργίες του ATT.

3.1.3.1 Error Response

Ένα error response στέλνεται από τον server στον client, ως αποτέλεσμα σφάλματος, το οποίο συνέβη κατά την επεξεργασία ενός request. Περιέχει το opcode της λειτουργίας και το handle του attribute, στα οποία συνέβη το σφάλμα, καθώς και ένα κωδικό σφάλματος που δηλώνει τον τύπο του. Στο πρότυπο, ορίζεται ένα σύνολο από κωδικούς σφαλμάτων, όπως, για παράδειγμα: Invalid Handle, Read Not Permitted, Write Not Permitted, Insufficient Authentication, Insufficient Authorization, Attribute Not Found, Attribute Not Long, Application Error.

Από τις ακόλουθες λειτουργίες, μόνο οι Write Command και Signed Write Command, που δεν έχουν response, καθώς και οι εκκινούμενες από τον server, Handle Value Notification και Handle Value Indication, δε χρησιμοποιούν Error Responses.

3.1.3.2 MTU Exchange

Ο client μπορεί να στείλει ένα Exchange MTU Request στον server για να τον ενημερώσει για το MTU που χρησιμοποιεί. Ο server απαντά με ένα Exchange MTU Response, στο οποίο ενημερώνει τον client για το δικό του MTU. Μετά την ολοκλήρωση της διαδικασίας, το MTU της σύνδεσης τίθεται στο ελάχιστο των δύο τιμών που ανταλλάχθηκαν. Το MTU Exchange δεν είναι απαραίτητο να συμβεί. Σε αυτή την περίπτωση το MTU, και στις δυο πλευρές, τίθεται στη default τιμή, δηλαδή σε 23.

3.1.3.3 Find Information Request – Response

Η δοσοληψία αυτή χρησιμοποιείται από τον client, προκειμένου να λάβει πληροφορίες σχετικά με τα attributes, που είναι αποθηκευμένα στον server. Συγκεκριμένα, ο client στέλνει ένα handle range και ο server απαντά στέλνοντας ζεύγη attribute handles και UUIDs, μέσω τον οποίων ο client δημιουργεί μία συσχέτιση μεταξύ attribute handles και attribute types. Τα SIG UUIDs των 16 bit στέλνονται σε διαφορετικά πακέτα από τα πλήρη UUIDs.

Αν τα αποτελέσματα δε χωράνε σε ένα πακέτο, στέλνονται μόνο όσα από αυτά χωράνε. Ο client πρέπει να στείλει νέο request για να λάβει τα επόμενα. Η διαδικασία αυτή συνεχίζεται μέχρι να τελειώσουν τα attributes που ζήτησε ο client, οπότε ο server του απαντά με Error Response τύπου Attribute Not Found. Η συγκεκριμένη τεχνική χρησιμοποιείται και σε άλλες τέτοιου τύπου δοσοληψίες. Στο αρχικό request, το handle range μπορεί να τεθεί σε 0x0001 ως 0xFFFF, ώστε να ανακαλυφθούν όλα τα πιθανά attributes.

3.1.3.4 Find By Type Value Request – Response

Η δοσοληψία αυτή χρησιμοποιείται από τον client, προκειμένου να αναζητήσει attributes σε ένα handle range, με συγκεκριμένο τύπο και τιμή, και να λάβει πληροφορίες σχετικά με τα handle groups που αυτά ορίζουν. Το UUID του τύπου μπορεί να είναι μόνο 16 bit, ενώ το attribute πρέπει να είναι τύπου grouping. Ο server επιστρέφει handle ranges για τα διάφορα groups που ικανοποιούν το request. Όπως και πριν, μπορεί να χρειαστούν περισσότερα του ενός requests, προκειμένου ο client να λάβει όλες τις πληροφορίες που θέλει.

3.1.3.5 Read By Type Request – Response

Η δοσοληψία αυτή χρησιμοποιείται από τον client, προκειμένου να αναζητήσει attributes σε ένα handle range, τα οποία είναι συγκεκριμένου τύπου, χωρίς να χρειάζεται να γνωρίζει τα handles αυτών. Το UUID του τύπου μπορεί να είναι σύντομο (16 bit) ή πλήρες (128 bit). Ο server επιστρέφει μία λίστα από ζεύγη handle και value για κάθε attribute του συγκεκριμένου τύπου. Όπως και πριν, μπορεί να χρειαστούν περισσότερα του ενός requests, προκειμένου ο client να λάβει όλες τις πληροφορίες που θέλει.

3.1.3.6 Read Request – Response

Η δοσοληψία αυτή χρησιμοποιείται από τον client, προκειμένου να διαβάσει την τιμή ενός attribute, παρέχοντας το handle αυτού. Ο server επιστρέφει την τιμή του attribute. Αν αυτή είναι μεγαλύτερη του (MTU - 1), τότε στέλνονται μόνο τα πρώτα (MTU - 1) bytes αυτής. Για να διαβαστούν όλα τα bytes χρησιμοποιείται η επόμενη δοσοληψία.

3.1.3.7 Read Blob Request – Response

Η δοσοληψία αυτή χρησιμοποιείται από τον client, προκειμένου να διαβάσει ένα κομμάτι της τιμής ενός attribute. Ο client παρέχει το handle του attribute και ένα offset μέσα στην τιμή του (με βάση το μηδέν), από το οποίο θα ξεκινήσει η αποστολή. Αν το attribute value μπορεί να σταλεί όλο σε ένα πακέτο, τότε ο server μπορεί να επιστρέψει σφάλμα με κωδικό "Attribute Not Long". Ο server επιστρέφει όσα bytes της τιμής, μετά το offset, μπορούν να χωρέσουν σε (MTU - 1) bytes. Η συγκεκριμένη δοσοληψία είναι ο μόνος τρόπος να διαβαστεί η τιμή ενός attribute με μέγεθος μεγαλύτερο από (MTU - 1) bytes.

3.1.3.8 Read Multiple Request – Response

Η δοσοληψία αυτή χρησιμοποιείται από τον client, προκειμένου να διαβάσει ταυτόχρονα πολλαπλές τιμές attributes. Ο client παρέχει μία λίστα από τουλάχιστον δύο handles για τα attributes που τον ενδιαφέρουν. Ο server απαντά με μία λίστα από attribute values. Η λίστα αυτή είναι η συνένωση (concatenation) των ζητούμενων τιμών. Αυτό σημαίνει ότι ο client πρέπει να γνωρίζει το μέγεθος κάθε τιμής, ώστε να μπορεί να τις διαχωρίσει. Επίσης, αν το μέγεθος της λίστας είναι μεγαλύτερο από (MTU - 1) bytes, αυτή κόβεται, και τα επιπλέον δεδομένα δεν αποστέλλονται.

3.1.3.9 Read By Group Type Request – Response

Η δοσοληψία αυτή χρησιμοποιείται από τον client, προκειμένου να διαβάσει τις τιμές attributes, για τα οποία γνωρίζει τον τύπο αλλά όχι το handle. Τα attributes αυτά πρέπει να είναι τύπου grouping. Ο client παρέχει ένα handle range αναζήτησης, καθώς και τον τύπο. Το UUID του τύπου μπορεί να είναι σύντομο (16 bit) ή πλήρες (128 bit). Η διαφορά από το Read By Type Request – Response είναι ότι, στην απάντηση του server, εκτός από το handle και την τιμή του κάθε attribute, περιέχεται και το τελευταίο handle του group, που ορίζεται από αυτό το grouping attribute. Έτσι, εφόσον κάθε group ξεκινά με το handle του grouping attribute, ο client ανακτά τόσο την τιμή του attribute, όσο και το group που αυτό ορίζει. Όπως και πριν, μπορεί να χρειαστούν περισσότερα του ενός requests, προκειμένου ο client να λάβει όλες τις πληροφορίες που θέλει.

3.1.3.10 Write Request – Response

Η δοσοληψία αυτή χρησιμοποιείται από τον client, προκειμένου να γράψει την τιμή ενός attribute. Ο client παρέχει το handle του attribute και τα δεδομένα που θέλει να γράψει στην τιμή του. Το μέγιστο μέγεθος των δεδομένων που μπορούν να γραφτούν με αυτή τη δοσοληψία είναι (MTU - 3) bytes. Για να γραφτούν περισσότερα bytes, πρέπει να χρησιμοποιηθεί μία σειρά από Prepare Write Requests – Responses, ακολουθούμενη από ένα Execute Write Request – Response. Αν δε συμβεί κάποιο σφάλμα, ο server επιστρέφει απλώς ένα πακέτο με το Write Response opcode, για να ενημερώσει τον client για την επιτυχία της εγγραφής.

3.1.3.11 Write Command

Η λειτουργία αυτή χρησιμοποιείται από τον client, προκειμένου να γράψει την τιμή ενός attribute. Όπως και παραπάνω, ο client παρέχει το handle του attribute και τα δεδομένα που θέλει να γράψει στην τιμή του. Το μέγιστο μέγεθος των δεδομένων που μπορούν να γραφτούν είναι επίσης (MTU - 3) bytes. Η διαφορά με το Write Request – Response είναι ότι ο server δε στέλνει καμία απάντηση, ανεξάρτητα αν η εγγραφή ολοκληρώθηκε επιτυχώς ή όχι.

3.1.3.12 Signed Write Command

Το Signed Write Command είναι μία παραλλαγή του Write Command, στην οποία χρησιμοποιείται το authentication signature του πακέτου. Ο client παρέχει και πάλι το handle του attribute και τα δεδομένα που θέλει να γράψει στην τιμή του. Αυτά συμπληρώνονται από τα 12 bytes του authentication signature, το οποίο υπολογίζεται πάνω στα υπόλοιπα τμήματα της PDU. Σε αυτή την περίπτωση το μέγιστο μέγεθος των δεδομένων που μπορούν να γραφτούν είναι (MTU - 15) bytes.

3.1.3.13 Prepare Write Request – Response, Execute Write Request – Response

Οι δοσοληψίες αυτές χρησιμοποιούνται από τον client, προκειμένου να ορίσει μια σειρά από εγγραφές στις τιμές πολλαπλών attributes, τις οποίες μπορεί στη συνέχεια να εκτελέσει όλες μαζί ως μία πράξη (atomic operation), ή να τις ακυρώσει. Για τη δημιουργία της σειράς εγγραφών, ο client εκτελεί μια σειρά από δοσοληψίες Prepare Write Request – Response, σε κάθε μια από τις οποίες παρέχει ένα attribute handle, ένα offset στην τιμή αυτού, και τα δεδομένα που θέλει να γράψει εκεί. Το μέγιστο μέγεθος των δεδομένων που μπορούν να γραφτούν με αυτή τη δοσοληψία είναι (MTU - 5) bytes. Ο server δε γράφει τα δεδομένα, αλλά τα τοποθετεί σε μία ουρά, μέχρι να γίνει η εκτέλεση ή η ακύρωση των εγγραφών. Ταυτόχρονα απαντά στον client, στέλνοντάς του πίσω το πακέτο που έλαβε, αλλάζοντας μόνο το opcode. Αυτό χρησιμεύει για επιβεβαίωση στον client, ότι ο server έλαβε τα σωστά δεδομένα. Αφού ο client ολοκληρώσει τον ορισμό της ουράς των εγγραφών, εκτελεί μια δοσοληψία Execute Write Request – Response, με την οποία δίνει εντολή στον server να εκτελέσει ή να ακυρώσει τις εγγραφές. Αν δε συμβεί κάποιο σφάλμα, ο server επιστρέφει απλώς ένα πακέτο με το Execute Write Response opcode, για να ενημερώσει τον client για την επιτυχία της εγγραφής.

Εξαιτίας του offset, οι συγκεκριμένες δοσοληψίες μπορούν να χρησιμοποιηθούν για να γραφτούν τιμές attributes, οι οποίες δε χωράνε σε ένα απλό Write Request. Επίσης, η επιβεβαίωση των δεδομένων από τον server, δίνει τη δυνατότητα εκτέλεσης αξιόπιστων εγγραφών (reliable writes).

3.1.3.14 Handle Value Notification

Με αυτή τη λειτουργία, ο server στέλνει στον client την τιμή ενός attribute, χωρίς επιβεβαίωση από τον δεύτερο για τη λήψη της. Ο server στέλνει το handle του attribute, ακολουθούμενο από την τιμή του. Το μέγιστο μέγεθος των δεδομένων που μπορούν να σταλούν με αυτή τη λειτουργία είναι (MTU - 3) bytes.

3.1.3.15 Handle Value Indication – Confirmation

Με αυτή τη δοσοληψία, ο server στέλνει στον client την τιμή ενός attribute, με επιβεβαίωση από τον δεύτερο για τη λήψη της. Όπως και στο Handle Value Notification, ο server στέλνει το handle του attribute, ακολουθούμενο από την τιμή του. Το μέγιστο μέγεθος των δεδομένων που μπορούν να σταλούν με αυτή τη δοσοληψία είναι επίσης (MTU - 3) bytes. Ο client απαντά με μία Handle Value Confirmation PDU.

3.2 Generic Attribute Profile

Το Generic Attribute Profile (GATT) χρησιμοποιεί τα attributes, που παρέχονται από το ATT, και τη δυνατότητα ομαδοποίησής τους, προκειμένου να δημιουργήσει μια ιεραρχία από δομές δεδομένων πάνω στην επίπεδη οργάνωση του ATT. Σε αυτή την υπο-ενότητα, περιγράφουμε πρώτα την ιεραρχία δομών που ορίζει το GATT. Στη συνέχεια, περιγράφουμε πώς το GATT δημιουργεί τις δομές αυτές πάνω στο ATT. Τέλος, περιγράφουμε τις βασικές λειτουργίες του GATT και το πώς αυτές συνδέονται με τις λειτουργίες του ATT.

Όπως στο ATT, έτσι και στο GATT, έχουμε δύο ρόλους, τον server και τον client. Ο ρόλος κάθε peer στο GATT είναι ο ίδιος με αυτόν στο ATT. Δηλαδή, ο GATT server είναι ATT server και ο GATT client είναι ATT client. Ο ρόλος μιας συσκευής στο GATT είναι ανεξάρτητος του ρόλου της στο GAP. Μια συσκευή μπορεί να είναι ταυτόχρονα GATT server και client, ή να εναλλάσσεται στους δύο ρόλους, ανεξάρτητα αν είναι peripheral ή central, και, αντίστοιχα, master ή slave της σύνδεσης.

3.2.1 Ιεραρχία του GATT

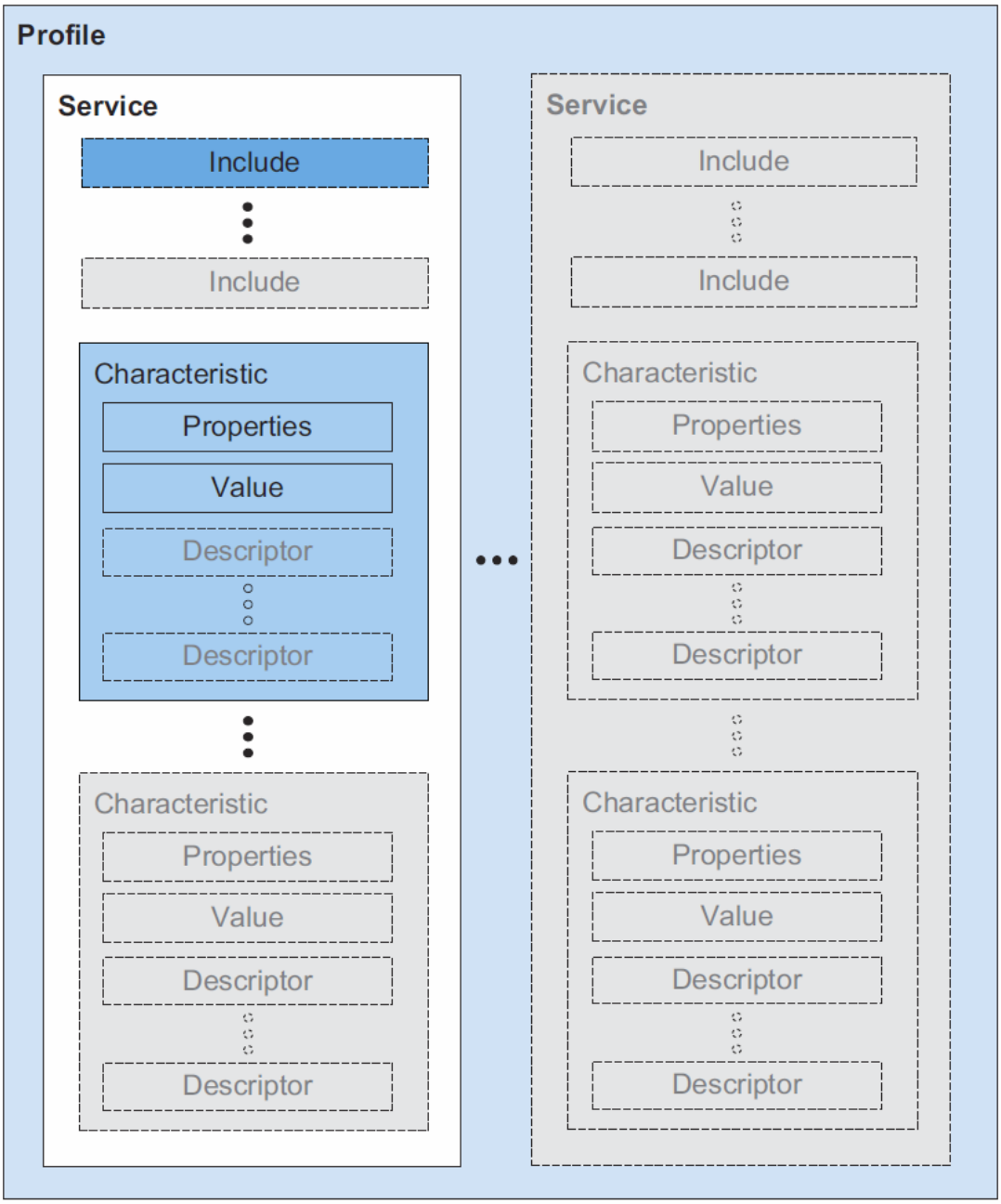
Στη ρίζα της ιεραρχίας του GATT βρίσκεται το προφίλ (profile). Ένα προφίλ αποτελείται από μία ή περισσότερες υπηρεσίες (services). Κάθε υπηρεσία αποτελείται από χαρακτηριστικά (characteristics), ενώ μπορεί να ορίζει σχέσεις με άλλες υπηρεσίες. Κάθε χαρακτηριστικό περιέχει μία τιμή, ενώ μπορεί, επίσης, να περιέχει και μετα-δεδομένα σχετικά με την τιμή του, με τη μορφή περιγραφέων (descriptors). Οι υπηρεσίες, τα χαρακτηριστικά και οι περιγραφείς αποθηκεύονται όλα σε attributes στον GATT server. Ο τρόπος με τον οποίο γίνεται αυτό περιγράφεται παρακάτω.

Η τιμές των χαρακτηριστικών είναι τα δεδομένα που ορίζονται από το GATT, και τα οποία χρησιμοποιούνται από την BLE εφαρμογή. Επίσης, με τη βοήθεια των control-point attributes, είναι δυνατόν να δημιουργηθούν και control-point χαρακτηριστικά, για την εκτέλεση λειτουργιών στον GATT server. Μια υπηρεσία είναι μια συλλογή τέτοιων δεδομένων, και συμπεριφορών που σχετίζονται με αυτά τα δεδομένα, με στόχο την επίτευξη μιας συγκεκριμένης λειτουργίας, που ορίζεται για αυτήν την υπηρεσία.

Οι υπηρεσίες μπορούν να ορίζουν σχέσεις με άλλες υπηρεσίες, κάνοντάς τις include. Μια υπηρεσία που γίνεται include από μία άλλη, γίνεται αναπόσπαστο κομμάτι αυτής. Δηλαδή όλα τα χαρακτηριστικά, οι περιγραφείς, και οι included υπηρεσίες της πρώτης εμφανίζονται πλέον και μέσα στη δεύτερη. Εδώ υπάρχει ο περιορισμός ότι μια υπηρεσία δεν μπορεί να μεταβάλει τη συμπεριφορά μιας άλλης, κάνοντάς την include. Με βάση τη σχέση include, έχουμε το διαχωρισμό των υπηρεσιών σε δύο τύπους: κύριες (primary) και δευτερεύουσες (secondary). Μια κύρια υπηρεσία παρέχει μία σημαντική λειτουργία της συσκευής και πρέπει να είναι ορατή στον χρήστη. Αντίθετα μια δευτερεύουσα υπηρεσία δεν παρέχει κάποια λειτουργία, που να σχετίζεται άμεσα με τη χρηστικότητα της συσκευής, αλλά υπάρχει μόνο για να γίνεται include από άλλες υπηρεσίες, παρέχοντας επιπλέον λειτουργικότητα σε αυτές. Μια κύρια υπηρεσία μπορεί επίσης να γίνεται include από άλλες υπηρεσίες. Σε αυτή την περίπτωση, το include μπορεί να γίνεται για λόγους επαύξησης της λειτουργικότητας μιας υπηρεσίας. Αυτό είναι χρήσιμο, επειδή μια υπηρεσία δεν πρέπει να μεταβάλλεται από τη στιγμή που θα γίνει διαθέσιμη, αφού, σε μια τέτοια περίπτωση, οι παλαιότεροι clients δε θα μπορούν να τη χρησιμοποιήσουν. Με το include, όμως, οι παλαιότεροι clients συνεχίζουν να βλέπουν την παλιά υπηρεσία, ενώ οι νεότεροι μπορούν να χρησιμοποιήσουν την αυξημένη λειτουργικότητα. Το include μπορεί επίσης να χρησιμοποιηθεί για την ομαδοποίηση άλλων υπηρεσιών σε μία καινούρια, ίσως και με προσθήκη επιπλέον λειτουργικότητας.

Αν δούμε την ιεραρχία του GATT με βάση το αντικειμενοστραφές μοντέλο, μία υπηρεσία είναι παρόμοια με μία κλάση, ενώ τα χαρακτηριστικά είναι τα δεδομένα που ενθυλακώνονται από αυτή. Η σχέση include μπορεί να ιδωθεί, είτε ως composition, όπου μία κλάση περιέχει δεδομένα τύπου μιας άλλης κλάσης, είτε ως inheritance, όπως στην περίπτωση της επαύξησης της λειτουργικότητας

Έχουμε, λοιπόν, ένα μοντέλο δεδομένων, όπου τα χαρακτηριστικά περιέχουν τα δεδομένα, και περιέχονται με τη σειρά τους σε υπηρεσίες, που τους δίνουν συγκεκριμένη συμπεριφορά και λειτουργικότητα. Στο ανώτερο επίπεδο, το προφίλ συνδυάζει μια σειρά από υπηρεσίες, προκειμένου να υλοποιήσει μια περίπτωση χρήσης.



3.2.2 Δομές Δεδομένων του GATT

Περιγράφουμε τώρα τον τρόπο με τον οποίο οι υπηρεσίες, τα χαρακτηριστικά και οι περιγραφείς, καθώς και οι σχέσεις μεταξύ υπηρεσιών, ορίζονται ως μια σειρά από attributes στην επίπεδη βάση δεδομένων του ATT.

3.2.2.1 Service Definition

Στο επίπεδο του ATT, ο ορισμός (definition) ενός service ξεκινά με ένα service declaration attribute, το οποίο ακολουθείται από μηδέν ή περισσότερα include definitions, ένα για κάθε service που γίνεται include από το αρχικό. Μετά τα include definitions, ακολουθούν τα characteristic definitions για τα χαρακτηριστικά του service. Η δομή αυτών των definitions περιγράφεται παρακάτω.

Το service declaration attribute είναι grouping attribute, οπότε δημιουργεί μία ομάδα από attributes, η οποία συνεχίζεται μέχρι το επόμενο service declaration attribute, όπου ξεκινά η επόμενη ομάδα και το επόμενο service. Όλα δηλαδή τα attributes ενός service definition ανήκουν στην ίδια ομάδα, με βάση την ομαδοποίηση των handles, όπως ορίζεται στο ATT.

Το service declaration attribute έχει την ακόλουθη μορφή:

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»	16-bit Bluetooth UUID or 128-bit UUID for Service	Read Only, No Authentication, No Authorization

Όπως βλέπουμε, ο τύπος του attribute είναι ένα 16-bit UUID, το οποίο ορίζεται από το SIG, και δηλώνει τον τύπο του service, αν δηλαδή είναι primary ή secondary. Η τιμή του attribute είναι το UUID της ίδιας της υπηρεσίας, με το οποίο η υπηρεσία μπορεί να αναγνωριστεί από συμβατούς clients. Το UUID αυτό μπορεί να είναι 16-bit, αν πρόκειται για υπηρεσία ορισμένη από το ίδιο το SIG, και 128-bit για υπηρεσίες ορισμένες από developers για τις δικές τους εφαρμογές.

Τα permissions του attribute επιτρέπουν σε οποιονδήποτε client να διαβάσει το service declaration. Το ίδιο ισχύει, όπως θα δούμε παρακάτω, και για τα υπόλοιπα attributes, που ορίζουν την ιεραρχία του GATT. Αυτό γίνεται επειδή τα attributes, που ορίζουν την ιεραρχία του GATT, δε θεωρούνται, ούτε περιέχουν, ευαίσθητες πληροφορίες. Είναι λοιπόν ορατά σε όλους τους clients, ώστε αυτοί να μπορούν να ανακτούν την ιεραρχία που ορίζεται στον server. Αν απαιτείται επιπλέον ασφάλεια για κάποια δεδομένα, αυτή ορίζεται στα συγκεκριμένα attributes, που αποθηκεύουν τις τιμές των δεδομένων αυτών.

3.2.2.2 Include Definition

Ένα include definition αποτελείται από ακριβώς ένα include declaration attribute, με την ακόλουθη μορφή:

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	0x2802 – UUID for «Include»	Included Service Attribute Handle	End Group Handle	Service UUID	Read Only, No Authentication, No Authorization

Ο τύπος του attribute είναι το 16-bit SIG UUID για το Include Declaration. Για τα permissions ισχύει ό,τι αναφέρθηκε παραπάνω. Η τιμή του attribute εξαρτάται από το αν το included service έχει 16-bit ή 128-bit UUID. Το handle range του included service υπάρχει πάντοτε, ενώ το UUID υπάρχει μόνο για services με 16-bit UUIDs.

3.2.2.3 Characteristic Definition

Ένα characteristic definition αποτελείται από ακριβώς ένα characteristic declaration attribute, το οποίο ακολουθείται από ακριβώς ένα characteristic value declaration attribute. Μετά από αυτό και πριν το characteristic declaration attribute του επόμενου χαρακτηριστικού, μπορούν να υπάρχουν μηδέν ή περισσότερα characteristic descriptor declaration attributes.

Το characteristic declaration attribute έχει την ακόλουθη μορφή:

Attribute Handle	Attribute Types	Attribute Value			Attribute Permissions
0xNNNN	0x2803–UUID for «Characteristic»	Charac-teristic Properties	Character-istic Value Attribute Handle	Character-istic UUID	Read Only, No Authentication, No Authorization

Ο τύπος του attribute είναι το 16-bit SIG UUID για το Characteristic Declaration. Για τα permissions ισχύει ό,τι αναφέρθηκε παραπάνω. Η τιμή του attribute αποτελείται από τρία στοιχεία. Το πρώτο είναι τα characteristic properties, ένα bitfield του ενός byte, το οποίο δηλώνει ποιες ενέργειες ορίζονται για το συγκεκριμένο χαρακτηριστικό. Οι πιθανές ενέργειες είναι οι εξής:

- Broadcast
- Read
- Write Without Response
- Write
- Notify
- Indicate
- Authenticated Signed Writes

Επίσης, υπάρχει ένα bit Extended Properties, το οποίο δηλώνει την παρουσία στο χαρακτηριστικό ενός descriptor, με επιπλέον properties.

Τα characteristic properties ακολουθούνται από το 16-bit handle του characteristic value attribute, το οποίο ακολουθεί το declaration. Τέλος, έχουμε το UUID, το οποίο ορίζει τον τύπο του χαρακτηριστικού. Όπως και στην περίπτωση του service, το UUID αυτό, χρησιμοποιείται από συμβατούς clients, προκειμένου να αναγνωρίσουν το χαρακτηριστικό. Το UUID μπορεί να είναι 16-bit, αν πρόκειται για χαρακτηριστικό ορισμένο από το ίδιο το SIG, και 128-bit για χαρακτηριστικά ορισμένα από developers για τις δικές τους εφαρμογές.

Το characteristic value attribute έχει την ακόλουθη μορφή:

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0xuuuu – 16-bit Bluetooth UUID or 128-bit UUID for Characteristic UUID	Characteristic Value	Higher layer profile or implementation specific

Η τιμή του handle του attribute πρέπει να είναι η ίδια με αυτή που περιέχεται στο characteristic declaration. Το τύπος του attribute είναι ο τύπος του χαρακτηριστικού, ίδιος με αυτόν που περιέχεται στο characteristic declaration.

Η τιμή του characteristic value attribute είναι η τιμή του χαρακτηριστικού. Σε αυτό δηλαδή το attribute αποθηκεύονται τα δεδομένα του profile, και αυτό το attribute διαβάζεται ή γράφεται από τους clients, όταν θέλουν να ανακτήσουν ή να μεταβάλουν τα δεδομένα ενός χαρακτηριστικού, όπως προβλέπεται στο profile. Για αυτόν τον λόγο, τα permissions καθορίζονται σε ανώτερο επίπεδο. Στην πράξη, εξαρτώνται από την εφαρμογή, που ορίζει το συγκεκριμένο χαρακτηριστικό.

3.2.2.4 Descriptor Declaration

Μετά από το characteristic value attribute, μπορούν να ακολουθούν μηδέν ή περισσότερα descriptor declaration attributes, καθένα από τα οποία δηλώνει έναν descriptor του χαρακτηριστικού. Όπως αναφέραμε παραπάνω, οι descriptors ορίζουν μετα-δεδομένα για την τιμή του χαρακτηριστικού. Για παράδειγμα, μπορεί να ορίζουν το format ή τη μονάδα μέτρησης της τιμής. Επίσης χρησιμοποιούνται για τον έλεγχο λειτουργιών του GATT server. Η σειρά των descriptor declaration attributes δεν έχει σημασία.

Το SIG έχει ορίσει μια σειρά από descriptors, με τύπους 16-bit SIG UUIDs, για κάποιες από τις πιο συνηθισμένες χρήσεις. Οι developers μπορούν να ορίσουν νέους descriptors για τις εφαρμογές τους, χρησιμοποιώντας 128-bit UUIDs για τους τύπους τους. Οι ορισμένοι από το SIG descriptors είναι οι ακόλουθοι:

- **Characteristic Extended Properties**: Ορίζει επιπλέον properties για το χαρακτηριστικό. Είναι υποχρεωτικό να υπάρχει, αν το Extended Properties bit έχει τεθεί στα properties του χαρακτηριστικού. Μόνο ένας descriptor αυτού του τύπου μπορεί να υπάρχει σε ένα χαρακτηριστικό.

- **Characteristic User Description**: Περιέχει κείμενο με την περιγραφή της τιμής του χαρακτηριστικού. Μόνο ένας descriptor αυτού του τύπου μπορεί να υπάρχει σε ένα χαρακτηριστικό.

- **Client Characteristic Configuration**: Χρησιμοποιείται για να καθορίσει κάποιες παραμέτρους λειτουργιών του GATT server. Πιο συγκεκριμένα, μέσω αυτού ο client ελέγχει την αποστολή notifications και indications από τον server. Η τιμή του είναι

ένα bitfield, που αρχικοποιείται σε μηδενική τιμή, κάτι που σημαίνει ότι, αρχικά, τα notifications και indications είναι απενεργοποιημένα. Ένας client μπορεί να τα ενεργοποιήσει, θέτοντας τα κατάλληλα bits στην τιμή. Σε αντίθεση με άλλες τιμές που δεν αναμένεται να διατηρούνται ανάμεσα σε συνδέσεις, το πρότυπο ορίζει ότι, σε bonded συσκευές, η τιμή του Client Characteristic Configuration πρέπει να διατηρείται και μετά από αποσύνδεση. Μάλιστα, η τιμή αυτή είναι διαφορετική ανά client. Δηλαδή, κάθε bonded client μπορεί να θέτει διαφορετικές ρυθμίσεις, και αυτές πρέπει να διατηρούνται και να επανέρχονται σε κάθε σύνδεση. Μόνο ένας descriptor αυτού του τύπου μπορεί να υπάρχει σε ένα χαρακτηριστικό.

- **Server Characteristic Configuration**: Χρησιμοποιείται επίσης για να καθορίσει κάποιες παραμέτρους λειτουργιών του GATT server. Πιο συγκεκριμένα, μέσω αυτού, ένας client μπορεί να ελέγξει την προσθήκη της τιμής του χαρακτηριστικού σε advertising πακέτα, που στέλνονται από τον server, αν αυτός είναι σε broadcast mode. Η τιμή του είναι ένα bitfield. Ένας client μπορεί να ενεργοποιήσει το broadcast, θέτοντας το κατάλληλο bit στην τιμή. Η τιμή αυτή είναι ίδια για όλους τους clients και διατηρείται ανεξαρτήτως συνδέσεων. Μόνο ένας descriptor αυτού του τύπου μπορεί να υπάρχει σε ένα χαρακτηριστικό.

- **Characteristic Presentation Format**: Ορίζει πληροφορίες σχετικά με το format της τιμής του χαρακτηριστικού. Μπορούν να υπάρχουν περισσότεροι από ένας descriptors αυτού του τύπου σε ένα χαρακτηριστικό. Αν υπάρχουν, όμως, πρέπει απαραίτητα να υπάρχει και ένας Characteristic Aggregate Format descriptor.

- **Characteristic Aggregate Format**: Αποτελείται από μία λίστα από handles, καθένα από τα οποία δείχνει σε ένα Characteristic Presentation Format descriptor, είτε στο ίδιο, είτε σε κάποιο άλλο χαρακτηριστικό. Χρησιμοποιείται σε περιπτώσεις που η τιμή του χαρακτηριστικού αποτελείται από περισσότερα του ενός πεδία, για να ορίσει το format της τιμής του κάθε πεδίου, μέσω των αντίστοιχων Presentation Format descriptors. Οπότε, η σειρά των handles των descriptors στη λίστα είναι σημαντική. Μόνο ένας descriptor αυτού του τύπου μπορεί να υπάρχει σε ένα χαρακτηριστικό.

3.2.2.4.1 Characteristic Extended Properties Declaration

Το Characteristic Extended Properties declaration attribute έχει την ακόλουθη μορφή:

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2900 – UUID for «Characteristic Extended Properties»	Characteristic Extended Properties Bit Field	Read Only, No Authentication, No Authorization

Η τιμή του είναι ένα 16-bit bitfield. Τα επιπλέον properties που ορίζονται σε αυτή είναι δύο:

- Reliable Write: Αν έχει τεθεί, τότε η λειτουργία Reliable Write είναι ενεργοποιημένη για αυτό το χαρακτηριστικό.
- Writable Auxiliaries: Αν έχει τεθεί, τότε το κείμενο, που περιέχεται στην τιμή του Characteristic User Description, μπορεί να μεταβληθεί από τον client.

3.2.2.4.2 Characteristic User Description Declaration

Το Characteristic User Description declaration attribute έχει την ακόλουθη μορφή:

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2901 – UUID for «Characteristic User Description»	Characteristic User Description UTF-8 String	Higher layer profile or implementation specific

3.2.2.4.3 Client Characteristic Configuration Declaration

Το Client Characteristic Configuration declaration attribute έχει την ακόλουθη μορφή:

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	02902 – UUID for «Client Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

Η τιμή του είναι ένα 16-bit bitfield. Τα δύο LSB της τιμής ενεργοποιούν τα notifications και indications.

- Bit 0 (0x0001): Ενεργοποιεί τα notifications για την τιμή του χαρακτηριστικού.
- Bit 1 (0x0002): Ενεργοποιεί τα indications για την τιμή του χαρακτηριστικού.

3.2.2.4.4 Server Characteristic Configuration Declaration

Το Server Characteristic Configuration declaration attribute έχει την ακόλουθη μορφή:

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2903 – UUID for «Server Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

Η τιμή του είναι ένα 16-bit bitfield. Το LSB της τιμής (bit 0, 0x0001) ενεργοποιεί το broadcast για την τιμή του χαρακτηριστικού.

3.2.2.4.5 Characteristic Presentation Format Declaration

Το Characteristic Presentation Format declaration attribute έχει την ακόλουθη μορφή:

Attribute Handle	Attribute Type	Attribute Value					Attribute Permissions
0xNNNN	0x2904 – UUID for «Characteristic Format»	Format	Exponent	Unit	Name Space	Description	Read only No Authentication, NO authorization

Η τιμή του έχει μέγεθος 7 bytes και αποτελείται από 5 πεδία. Το πεδίο format, του ενός byte, καθορίζει τον τύπο της τιμής του χαρακτηριστικού. Το πρότυπο ορίζει μια σειρά από δυνατές τιμές και τους αντίστοιχους τύπους δεδομένους. Οι δυνατοί τύποι είναι οι εξής:

boolean	1 bit
unsigned integer	2, 4, 8, 16, 24, 32, 48, 64, 128 bit
singed integer	8, 16, 24, 32, 48, 64, 128 bit
floating point	IEEE-754 32/64 bit floating point
SFLOAT	IEEE-11073 16 bit SFLOAT
FLOAT	IEEE-11073 32 bit FLOAT
duint16	IEEE-20601 format
string	UTF-8/UTF-16 string
struct	Opaque structure

Το πεδίο Exponent, του ενός byte, ορίζει τη δύναμη του 10 με την οποία πρέπει να πολλαπλασιαστεί η τιμή του χαρακτηριστικού, ώστε να πάρουμε την πραγματική του τιμή. Το πεδίο Unit είναι ένα 16-bit UUID, το οποίο ορίζει τη μονάδα μέτρησης της τιμής του χαρακτηριστικού. Το SIG έχει ορίσει πολλές δυνατές τιμές για αυτό το UUID, στα "Assigned Numbers", τα οποία διατηρεί. Τα πεδία Namespace (1 byte) και Description (2 bytes) επιτρέπουν σε οργανισμούς, αναγνωρισμένους από το SIG, να κωδικοποιήσουν επιπλέον περιγραφές για το χαρακτηριστικό.

3.2.2.4.6 Characteristic Aggregate Format Declaration

Το Characteristic Aggregate Format declaration attribute έχει την ακόλουθη μορφή:

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2905 – UUID for «Characteristic Aggregate Format»	List of *Attribute Handles* for the Characteristic Presentation Format Declarations	Read only No authentication No authorization

3.2.2.5 Παράδειγμα ομαδοποίησης attributes

Ακολουθεί ένα παράδειγμα ομαδοποίησης attributes από το GATT. Έστω ότι έχουμε το ακόλουθο attribute table στο επίπεδο του ATT.

Handle	Type	Value
0x0001	«Primary Service»	«GAP Service»
0x0004	«Characteristic»	{0x02, 0x0006, «Device Name»}
0x0006	«Device Name»	"Example Device"
0x0008	«Characteristic»	{0x02, 0x0009, «Appearance»}
0x0009	«Appearance»	«Thermometer»
0x0201	«Primary Service»	«Thermometer Service»
0x0203	«Characteristic»	{0x12, 0x0204, «Temperature»}
0x0204	«Temperature»	0x051F
0x0205	«Presentation Format»	{0x0E, 0xFE, «Celsius», «SIG», «Outside»}

Όπως παρατηρούμε, έχουμε δύο service definitions, τα οποία ορίζουν τα handle ranges (0x0001-0x0009) και (0x0201-0x0205). Τα service declarations βρίσκονται στα attributes 0x0001 και 0x0201. Το πρώτο service definition περιέχει δύο characteristic definitions, που ξεκινούν στα attributes 0x0004 και 0x0008. Το δεύτερο περιέχει ένα characteristic definition, που ξεκινά στο attribute 0x0203, και το οποίο με τη σειρά του, περιέχει ένα descriptor declaration στο attribute 0x0205. Καθένα από τα characteristic definitions ξεκινά με το characteristic declaration, το οποίο ακολουθείται από το characteristic value declaration. Έτσι, με βάση όσα αναφέραμε παραπάνω, έχουμε την ακόλουθη ιεραρχία GATT.

Handle	Type	Value
Service		
0x0001	«Primary Service»	«GAP Service»
Characteristic		
0x0004	«Characteristic»	{0x02, 0x0006, «Device Name»}
0x0006	«Device Name»	"Example Device"
Characteristic		
0x0008	«Characteristic»	{0x02, 0x0009, «Appearance»}
0x0009	«Appearance»	«Thermometer»
Service		
0x0201	«Primary Service»	«Thermometer Service»
Characteristic		
0x0203	«Characteristic»	{0x12, 0x0204, «Temperature»}
0x0204	«Temperature»	0x051F
0x0205	«Presentation Format»	{0x0E, 0xFE, «Celsius», «SIG», «Outside»}

3.2.3 Λειτουργίες του GATT

Το GATT ορίζει ένα σύνολο από λειτουργίες, με τις οποίες ένας client μπορεί να ανακαλύψει την ιεραρχία δεδομένων, που είναι αποθηκευμένη στον server, και, στη συνέχεια, να ανακτήσει ή να μεταβάλει αυτά τα δεδομένα. Επίσης, ορίζει λειτουργίες με τις οποίες ο server μπορεί να στείλει δεδομένα στον client. Κάθε λειτουργία του GATT επιτυγχάνεται με τη βοήθεια μιας ή περισσότερων λειτουργιών του ATT.

Οι λειτουργίες του GATT χωρίζονται σε 11 κατηγορίες (features), καθεμία από τις οποίες περιέχει ένα ή περισσότερα sub-procedures. Τα features είναι τα ακόλουθα:

1. Server Configuration
2. Primary Service Discovery
3. Relationship Discovery
4. Characteristic Discovery
5. Characteristic Descriptor Discovery
6. Characteristic Value Read
7. Characteristic Value Write
8. Characteristic Value Notification
9. Characteristic Value Indication
10. Characteristic Descriptor Value Read
11. Characteristic Descriptor Value Write

3.2.3.1 Server Configuration

Το Server Configuration feature περιέχει ένα sub-procedure, το **Exchange MTU**. Η λειτουργία είναι όμοια με την αντίστοιχη του ATT.

3.2.3.2 Primary Service Discovery

Το Primary Service Discovery feature περιέχει δύο sub-procedures:

- **Discover All Primary Services**: Χρησιμοποιείται από τον client, προκειμένου να ανακαλύψει όλα τα primary services που υπάρχουν στον server. Για τον σκοπό αυτό, ο client εκτελεί μια σειρά από δοσοληψίες Read By Group Type Request – Response, θέτοντας ως type το SIG UUID “Primary Service”, με τις οποίες ανακτά από τον server τα UUIDs των primary services και τα handle ranges των group που αυτές ορίζουν. Τα handle ranges αυτά, αντιστοιχούν στο service definition της κάθε υπηρεσίας.

- **Discover Primary Services By Service UUID**: Χρησιμοποιείται από τον client, προκειμένου να ανακαλύψει μία συγκεκριμένη υπηρεσία στον server, της οποίας γνωρίζει το UUID. Για τον σκοπό αυτό, ο client εκτελεί μια σειρά από δοσοληψίες Find By Type Value Request – Response, θέτοντας ως type το SIG UUID “Primary Service” και ως value το UUID της υπηρεσίας που τον ενδιαφέρει. Ο server απαντά με τα handle ranges των group, που ορίζονται από τα primary services με το συγκεκριμένο UUID. Τα handle ranges αυτά, αντιστοιχούν στο service definition της κάθε υπηρεσίας.

3.2.3.3 Relationship Discovery

Το Relationship Discovery feature περιέχει ένα sub-procedure (το Service Discovery πρέπει να έχει γίνει ήδη, οπότε είναι γνωστά τα handle ranges των service definitions):

- **Find Included Services**: Χρησιμοποιείται από τον client, προκειμένου να ανακαλύψει τα services, που γίνονται included από ένα service. Για τον σκοπό αυτό, ο client εκτελεί μια σειρά από δοσοληψίες Read By Type Request – Response, θέτοντας ως type το SIG UUID "Include" και ως handle range αυτό του service που τον ενδιαφέρει, με τις οποίες ανακτά από τον server τα handle ranges των included services, που περιέχονται στο συγκεκριμένο service definition. Στην περίπτωση που πρόκειται για services με 16-bit UUID, επιστρέφεται και το UUID.

3.2.3.4 Characteristic Discovery

Το Characteristic Discovery feature περιέχει δύο sub-procedures (το Service Discovery πρέπει να έχει γίνει ήδη, οπότε είναι γνωστά τα handle ranges των service definitions):

- **Discover All Characteristic of a Service**: Χρησιμοποιείται από τον client, προκειμένου να ανακαλύψει όλα τα χαρακτηριστικά ενός service. Για τον σκοπό αυτό, ο client εκτελεί μια σειρά από δοσοληψίες Read By Type Request – Response, θέτοντας ως type το SIG UUID "Characteristic" και ως handle range αυτό του service που τον ενδιαφέρει, με τις οποίες ανακτά από τον server τις τιμές των characteristic declaration attributes που περιέχονται στο συγκεκριμένο service definition. Αυτές περιέχουν τόσο τα UUIDs των χαρακτηριστικών, όσο και τα handles των characteristic value attributes.

- **Discover Characteristic by UUID**: Χρησιμοποιείται από τον client, προκειμένου να ανακαλύψει χαρακτηριστικά ενός service με συγκεκριμένο UUID. Η διαδικασία είναι όμοια με την προηγούμενη, με τη διαφορά ότι τα χαρακτηριστικά που ανακαλύπτονται, ελέγχονται στην πλευρά του client για το ζητούμενο UUID.

3.2.3.5 Characteristic Descriptor Discovery

Το Characteristic Discovery feature περιέχει ένα sub-procedure:

- **Discover All Characteristic Descriptors**: Χρησιμοποιείται από τον client, προκειμένου να ανακαλύψει όλους τους descriptors ενός χαρακτηριστικού. Για τον σκοπό αυτό, ο client εκτελεί μια σειρά από δοσοληψίες Find Information Request – Response, πάνω στα handle ranges που έχουν προκύψει για κάθε χαρακτηριστικό από τις προηγούμενες διαδικασίες.

3.2.3.6 Characteristic Value Read

Το Characteristic Value Read feature περιέχει τέσσερα sub-procedures (η ανακάλυψη των χαρακτηριστικών πρέπει να έχει γίνει ήδη, οπότε είναι γνωστά τα handles, εκτός της περίπτωσης του Read Using Characteristic UUID, στην οποία δεν είναι απαραίτητο):

- **Read Characteristic Value**: Χρησιμοποιείται από τον client, προκειμένου να διαβάσει την τιμή ενός χαρακτηριστικού. Για τον σκοπό αυτό, ο client εκτελεί μια δοσοληψία Read Request – Response.

- **Read Using Characteristic UUID**: Χρησιμοποιείται από τον client, προκειμένου να διαβάσει την τιμή ενός χαρακτηριστικού με συγκεκριμένο UUID. Για τον σκοπό αυτό, ο client εκτελεί μια δοσοληψία Read By Type Request – Response, με type ίσο με το UUID. Το sub-procedure αυτό χρησιμοποιείται συνήθως σε περιπτώσεις χαρακτηριστικών με γνωστό UUID, τα οποία είναι μοναδικά στον server.

- **Read Long Characteristic Values**: Χρησιμοποιείται από τον client, προκειμένου να διαβάσει την τιμή ενός χαρακτηριστικού, όταν αυτή δεν μπορεί να σταλεί σε ένα πακέτο. Για τον σκοπό αυτό, ο client εκτελεί μια σειρά από δοσοληψίες Read Blob Request – Response.

- **Read Multiple Characteristic Values**: Χρησιμοποιείται από τον client, προκειμένου να διαβάσει τις τιμές πολλαπλών χαρακτηριστικών. Για τον σκοπό αυτό, ο client εκτελεί μια δοσοληψία Read Multiple Request – Response.

3.2.3.7 Characteristic Value Write

Το Characteristic Value Write feature περιέχει πέντε sub-procedures (η ανακάλυψη των χαρακτηριστικών πρέπει να έχει γίνει ήδη, οπότε είναι γνωστά τα handles):

- **Write Characteristic Value**: Χρησιμοποιείται από τον client, προκειμένου να γράψει την τιμή ενός χαρακτηριστικού. Για τον σκοπό αυτό, ο client εκτελεί μια δοσοληψία Write Request – Response.

- **Write Without Response**: Χρησιμοποιείται από τον client, προκειμένου να γράψει την τιμή ενός χαρακτηριστικού, χωρίς να περιμένει απάντηση από τον server. Για τον σκοπό αυτό, ο client εκτελεί τη λειτουργία Write Command.

- **Signed Write Without Response**: Χρησιμοποιείται από τον client, προκειμένου να γράψει την τιμή ενός χαρακτηριστικού, με ενεργοποιημένη την πιστοποίηση αυθεντικότητας, χωρίς να περιμένει απάντηση από τον server. Για τον σκοπό αυτό, ο client εκτελεί τη λειτουργία Signed Write Command.

- **Write Long Characteristic Values**: Χρησιμοποιείται από τον client, προκειμένου να γράψει την τιμή ενός χαρακτηριστικού, όταν τα νέα δεδομένα της τιμής δεν είναι δυνατόν να σταλούν σε ένα πακέτο. Για τον σκοπό αυτό, ο client εκτελεί μια σειρά από δοσοληψίες Prepare Write Request – Response, ακολουθούμενες από μια δοσοληψία Execute Write Request – Response.

- **Characteristic Value Reliable Writes**: Χρησιμοποιείται από τον client, προκειμένου να γράψει την τιμή ενός ή περισσότερων χαρακτηριστικών αξιόπιστα. Για τον σκοπό αυτό, ο client εκτελεί μια σειρά από δοσοληψίες Prepare Write Request – Response, ακολουθούμενες από μια δοσοληψία Execute Write Request – Response.

3.2.3.8 Characteristic Value Notification

Το Characteristic Value Notification feature περιέχει ένα sub-procedure:

- **Notifications**: Χρησιμοποιείται από τον server, προκειμένου να στείλει την τιμή ενός χαρακτηριστικού στον client, χωρίς επιβεβαίωση για τη λήψη του. Ο server εκτελεί τη λειτουργία Handle Value Notification.

3.2.3.9 Characteristic Value Indication

Το Characteristic Value Indication feature περιέχει ένα sub-procedure:

- **Indications**: Χρησιμοποιείται από τον server, προκειμένου να στείλει την τιμή ενός χαρακτηριστικού στον client, με επιβεβαίωση για τη λήψη του. Ο server εκτελεί τη δοσοληψία Handle Value Indication – Confirmation.

3.2.3.10 Characteristic Descriptor Value Read

Το Characteristic Descriptor Value Read feature περιέχει δύο sub-procedures (η ανακάλυψη των descriptors πρέπει να έχει γίνει ήδη, οπότε είναι γνωστά τα handles):

- **Read Characteristic Descriptors**: Χρησιμοποιείται από τον client, προκειμένου να διαβάσει την τιμή ενός descriptor. Για τον σκοπό αυτό, ο client εκτελεί μια δοσοληψία Read Request – Response.

- **Read Long Characteristic Descriptors**: Χρησιμοποιείται από τον client, προκειμένου να διαβάσει την τιμή ενός descriptor, όταν αυτή δεν μπορεί να σταλεί σε ένα πακέτο. Για τον σκοπό αυτό, ο client εκτελεί μια σειρά από δοσοληψίες Read Blob Request – Response.

3.2.3.11 Characteristic Descriptor Value Write

Το Characteristic Descriptor Value Write feature περιέχει δύο sub-procedures (η ανακάλυψη των descriptors πρέπει να έχει γίνει ήδη, οπότε είναι γνωστά τα handles):

- **Write Characteristic Descriptors**: Χρησιμοποιείται από τον client, προκειμένου να γράψει την τιμή ενός descriptor. Για τον σκοπό αυτό, ο client εκτελεί μια δοσοληψία Write Request – Response.

- **Write Long Characteristic Descriptors**: Χρησιμοποιείται από τον client, προκειμένου να γράψει την τιμή ενός descriptor, όταν τα νέα δεδομένα της τιμής δεν είναι δυνατόν να σταλούν σε ένα πακέτο. Για τον σκοπό αυτό, ο client εκτελεί μια σειρά από δοσοληψίες Prepare Write Request – Response, ακολουθούμενες από μια δοσοληψία Execute Write Request – Response.

3.3 GATT – GAP services

Τα GATT και GAP ορίζουν δικά τους GATT services, τα οποία παρέχουν πληροφορίες για τη συσκευή, ή βοηθούν στην εκτέλεση λειτουργιών του πρωτοκόλλου.

3.3.1 GATT Service

Το GATT service περιέχει ένα μόνο χαρακτηριστικό, το Service Changed, του οποίου η τιμή μπορεί μόνο να γίνει indicate. Η χρήση του προορίζεται για τις περιπτώσεις εκείνες, στις οποίες μπορούν να συμβούν αλλαγές στα υπόλοιπα GATT services, που υπάρχουν στον server, και, έχει άμεση σχέση με τη δυνατότητα attribute caching. Η τιμή του χαρακτηριστικού Service Changed περιέχει ένα handle range. Ο server, λοιπόν, μπορεί να ειδοποιήσει έναν client για τις αλλαγές στις υπηρεσίες του, στέλνοντάς του (indication) το handle range που έχει επηρεαστεί και δε θεωρείται πλέον έγκυρο. Ο client, στη συνέχεια, αφαιρεί τα συγκεκριμένα handles από την cache του, και προχωρά σε νέο service discovery. Αυτή η λειτουργία είναι σημαντική, γιατί διαφορετικά το attribute caching δε θα μπορούσε να λειτουργήσει εκτός συνδέσεων, αφού οι clients δε θα γνώριζαν αν τα τοπικά αποθηκευμένα handles ισχύουν ακόμα. Έτσι, θα έπρεπε να εκτελέσουν ξανά όλη τη διαδικασία του discovery, με σημαντικό κόστος επικοινωνίας.

Η ύπαρξη ή μη του GATT service δίνει τη δυνατότητα στους clients να ελέγξουν αν υπάρχει περίπτωση αλλαγής των υπολοίπων services, που περιέχονται στον server. Αν το GATT service δεν υπάρχει, τότε ένας client μπορεί να θεωρήσει ότι τα services στον συγκεκριμένο server δεν πρόκειται να αλλάξουν, οπότε το attribute caching μεταξύ συνδέσεων είναι έγκυρο. Αν υπάρχει το GATT service, τότε η συμπεριφορά των clients εξαρτάται από το αν είναι bonded. Σε bonded clients, αν τα services μεταβληθούν εκτός σύνδεσης, τότε, με τη νέα σύνδεση του client, ο server στέλνει άμεσα το Service Changed indication. Μάλιστα, ο server πρέπει να θυμάται ότι ενημέρωσε έναν client για τις αλλαγές, ώστε να μην του ξαναστείλει το ίδιο indication. Σε μη bonded clients, όμως, δεν είναι δυνατόν να γίνει κάτι τέτοιο. Άρα, οι μη bonded clients, σε περίπτωση ύπαρξης του GATT service, πρέπει απαραίτητα να κάνουν νέο service discovery σε κάθε σύνδεση.

3.3.2 GAP Service

Το GAP service πρέπει να υπάρχει υποχρεωτικά σε συσκευές BLE με GAP ρόλους peripheral ή central. Στην έκδοση 4.0 του Bluetooth, το GAP service περιείχε 5 χαρακτηριστικά. Τα δύο από αυτά καταργήθηκαν στην έκδοση 4.1. Μόνο ένα χαρακτηριστικό των τύπων αυτών μπορεί να υπάρχει σε κάθε συσκευή. Τα χαρακτηριστικά του GAP service είναι τα ακόλουθα:

- **Device Name**: Η τιμή του είναι ένα UTF-8 string, το οποίο περιέχει το Bluetooth name της συσκευής. Χρησιμοποιείται κατά τη διαδικασία name discovery του GAP, κατά την οποία διαβάζεται χωρίς να χρειάζεται να γίνει πρώτα service discovery, εκτελώντας τη λειτουργία Read Using Characteristic UUID του GATT. Είναι υποχρεωτικό και σε peripheral και σε central συσκευές.

- **Appearance**: Η τιμή του είναι ένα αναγνωριστικό 16-bit, οι δυνατές τιμές του οποίου ορίζονται στα SIG Assigned Numbers. Υπάρχουν για παράδειγμα τιμές για “Generic Phone”, “Generic Computer”, “Generic Media Player”, “Generic Watch”, “Keyboard”, “Mouse” κλπ. Η χρησιμότητά του είναι κυρίως για λόγους παρουσίασης της συσκευής στον χρήστη, μέσω κάποιου user interface. Για παράδειγμα, μετά την

αναζήτηση συσκευών, αυτές παρουσιάζονται σε μια λίστα, με εικονίδια δίπλα στο όνομά τους, τα οποία εξαρτώνται από τον τύπο που αυτές δηλώνουν στο Appearance. Είναι υποχρεωτικό και σε peripheral και σε central συσκευές.

- **Peripheral Preferred Connection Parameters**: Αυτό το προαιρετικό χαρακτηριστικό, μπορεί να χρησιμοποιηθεί από ένα peripheral, προκειμένου να δηλώσει στο central, με το οποίο συνδέεται, τις επιθυμητές για αυτό παραμέτρους σύνδεσης. Το central είναι αυτό που, ως master της σύνδεσης, ελέγχει τις παραμέτρους της. Το peripheral από την άλλη, μπορεί να επιθυμεί συγκεκριμένες παραμέτρους σύνδεσης, βελτιστοποιημένες ως προς τη λειτουργία του. Διαβάζοντας, λοιπόν, το χαρακτηριστικό αυτό, το central μπορεί, στη συνέχεια, να θέσει τις συγκεκριμένες παραμέτρους. Αυτό, όμως, δεν είναι υποχρεωτικό.

- **Peripheral Privacy Flag (4.0 only)**: Η τιμή του είναι 1 byte, το LSB του οποίου καθορίζει αν το privacy είναι ενεργοποιημένο για τη συγκεκριμένη peripheral συσκευή. Αν το χαρακτηριστικό δεν υπάρχει, τότε θεωρείται ότι η συσκευή δεν υποστηρίζει privacy. Αν υπάρχει και είναι ενεργοποιημένο, τότε η διεύθυνση, που περιέχεται στα advertising πακέτα της συσκευής, είναι private, διαφορετικά είναι η public διεύθυνση της συσκευής. Με αυτόν τον τρόπο, ένα peripheral μπορεί να αποκρύπτει την public διεύθυνση του, για την αποφυγή της παρακολούθησής του από κακόβουλους χρήστες. Το central μπορεί να μεταβάλλει την τιμή αυτή, δεν είναι, όμως, σίγουρο ότι η εγγραφή θα επιτύχει. Αν το Reconnection Address υπάρχει και το privacy είναι ενεργοποιημένο, τότε η διεύθυνση, που περιέχεται στο Reconnection Address, είναι αυτή η οποία χρησιμοποιείται στα advertising πακέτα.

- **Reconnection Address (4.0 only)**: Η τιμή του είναι μια non-resolvable private διεύθυνση Bluetooth, η οποία, σε συνδυασμό με το Privacy Flag, μπορεί να χρησιμοποιηθεί για την υποστήριξη privacy σε μια peripheral συσκευή. Όπως αναφέρθηκε παραπάνω, το peripheral χρησιμοποιεί αυτή τη διεύθυνση κατά το advertising. Έτσι, οι μόνες συσκευές, που μπορούν να αναγνωρίσουν το peripheral, είναι όσες είναι bonded με αυτό και γνωρίζουν το Reconnection Address. Άλλες συσκευές δεν μπορούν να κάνουν την αντιστοίχηση μεταξύ του non-resolvable private address και του peripheral, οπότε δεν μπορούν να παρακολουθούν τη συσκευή. Η χρησιμότητα του Reconnection Address είναι ότι επιτρέπει στις central συσκευές να μη χρειάζεται να ελέγχουν, μέσω των IRKs, όλα τα advertising πακέτα, για resolvable private addresses, που να ταιριάζουν στο peripheral, το οποίο αναζητούν. Εφόσον γνωρίζουν το Reconnection Address, μπορούν να το τοποθετήσουν στη white list. Βέβαια, για να λειτουργήσει το privacy με αυτόν τον τρόπο, πρέπει το Reconnection Address να αλλάζει συχνά, ακόμα και σε κάθε σύνδεση, για αυτό και δίνεται η δυνατότητα στο central να μεταβάλλει την τιμή του.

Τα δύο τελευταία χαρακτηριστικά είναι δυνατόν να δημιουργήσουν προβλήματα στην επικοινωνία μεταξύ peripheral και central. Για παράδειγμα, αν το peripheral είναι bonded με πολλά centrals και ένα από αυτά αποφασίσει να αλλάξει το Privacy Flag, τότε, τα υπόλοιπα δεν μπορούν πλέον να συνδεθούν με το peripheral, αφού η διεύθυνση που χρησιμοποιεί το τελευταίο είναι διαφορετική από αυτή που αναμένουν. Εξαιτίας αυτών των προβλημάτων, και της μικρής χρήσης των χαρακτηριστικών αυτών στην πράξη, αφαιρέθηκαν από το πρότυπο στην έκδοση 4.1.

3.4 GATT-based profiles

Είδαμε παραπάνω πώς το GATT χρησιμοποιεί τα attributes του ATT, ώστε να δημιουργήσει μια ιεραρχία δεδομένων πάνω από την επίπεδη οργάνωση του ATT. Μέσω της ιεραρχίας αυτής, μπορούν να οριστούν τα δεδομένα κάθε εφαρμογής και περίπτωσης χρήσης. Όπως είδαμε, τα δεδομένα αποθηκεύονται στις τιμές των χαρακτηριστικών, τα οποία ο GATT client μπορεί να ανακτήσει ή να μεταβάλει, χρησιμοποιώντας τις παρεχόμενες από το GATT λειτουργίες. Επίσης, μπορεί, μέσω των control-point χαρακτηριστικών, να εκκινήσει συγκεκριμένες λειτουργίες στον server. Τα χαρακτηριστικά περιέχονται μέσα σε υπηρεσίες, οι οποίες τους προσδίδουν συμπεριφορά, ώστε να δημιουργείται μια συγκεκριμένη λειτουργικότητα. Στο ανώτερο επίπεδο, το προφίλ χρησιμοποιεί ένα σύνολο από υπηρεσίες, προκειμένου να υλοποιήσει μια περίπτωση χρήσης. Με άλλα λόγια, το προφίλ αποτελεί τον τυπικό ορισμό του επιπέδου εφαρμογής. Είναι το πρότυπο που πρέπει να υλοποιείται από τις συμβατές συσκευές (servers και clients), προκειμένου αυτές να μπορούν να ανταλλάξουν τα δεδομένα που ορίζονται στο προφίλ, και να εκτελέσουν τις διαδικασίες που προβλέπονται για τη συγκεκριμένη περίπτωση χρήσης.

3.4.1 Τυπικός ορισμός – XML Schemas

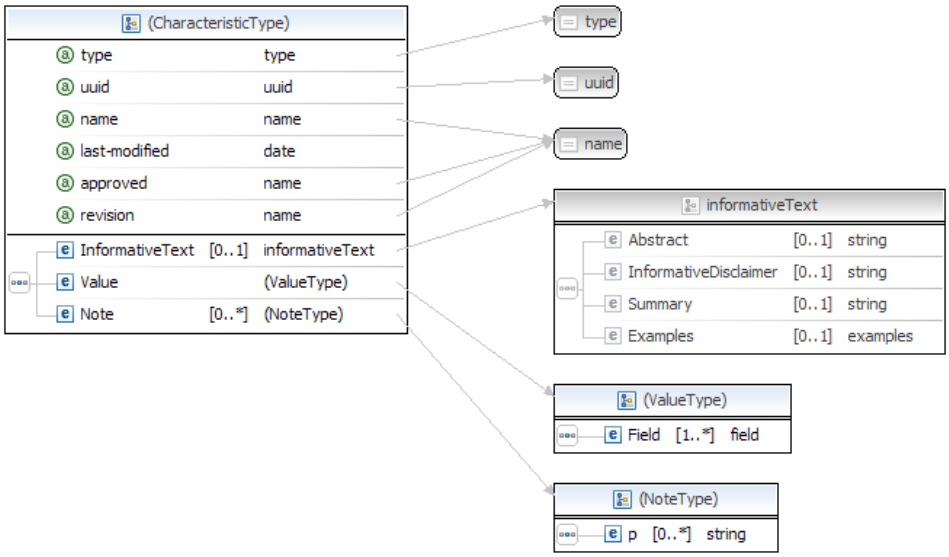
Το Bluetooth SIG έχει ορίσει μια σειρά από XML schemas, με τη βοήθεια των οποίων μπορούν να οριστούν τυπικά, τα χαρακτηριστικά, οι υπηρεσίες και τα ίδια τα προφίλ.

Για ένα χαρακτηριστικό, ορίζεται το όνομα, ο τύπος και η μορφή της τιμής του. Για παράδειγμα μπορούν να οριστούν πεδία συγκεκριμένων μεγεθών, υποχρεωτικά ή προαιρετικά, και να καθοριστούν οι δυνατές τιμές και η σημασία τους. Για μια υπηρεσία, ορίζεται το όνομα, ο τύπος, τυχόν εξαρτήσεις από άλλες υπηρεσίες, καθώς και σχέσεις (include) με άλλες υπηρεσίες. Επίσης, ορίζονται τα χαρακτηριστικά από τα οποία αυτή αποτελείται, υποχρεωτικά ή προαιρετικά, μαζί με τα properties, όπου δηλώνονται οι GATT λειτουργίες που υποστηρίζονται, και τους descriptors που πρέπει να ορίζονται για το καθένα. Ο τύπος ενός χαρακτηριστικού ή μιας υπηρεσίας ορίζεται μέσω ενός ονόματος τύπου, το οποίο ακολουθεί παρόμοιο naming convention με αυτό που χρησιμοποιείται στα Java packages (π.χ. org.bluetooth.service.glucose, org.bluetooth.characteristic.battery_level). Μέσω της XML, ορίζεται και το αναγνωριστικό UUID του κάθε τύπου.

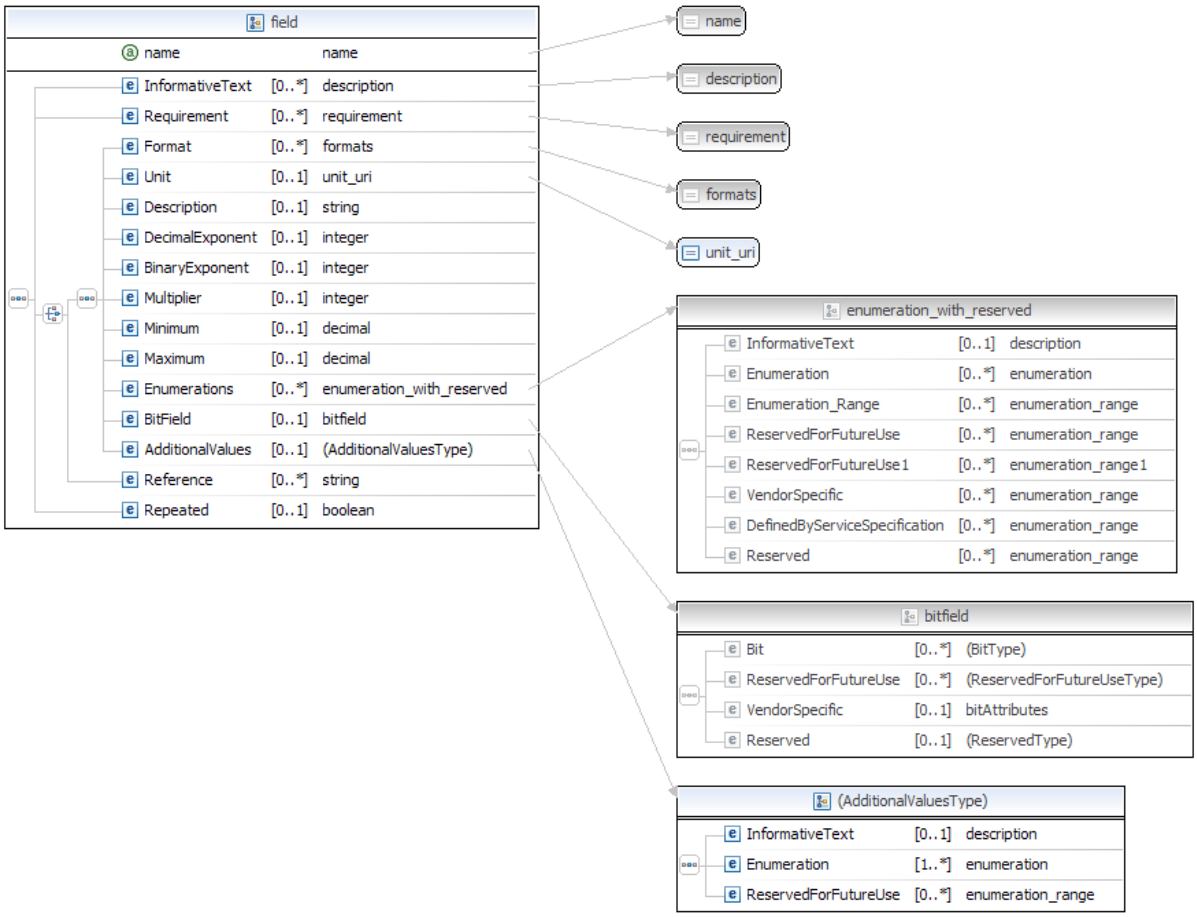
Το προφίλ ορίζει τους ρόλους, τουλάχιστον δύο, που αφορούν τη συγκεκριμένη περίπτωση χρήσης. Κάθε ρόλος μπορεί να είναι GATT server ή/και GATT client (στην πράξη συνήθως μόνο το ένα). Μέσω της XML, ορίζονται οι υπηρεσίες που πρέπει να υποστηρίζει ο κάθε ρόλος, υποχρεωτικά ή προαιρετικά.

Αυτό που απουσιάζει από τον τυπικό ορισμό είναι η συμπεριφορά. Αυτή πρέπει να οριστεί ανεξάρτητα. Στην περίπτωση των adopted profiles του SIG, υπάρχει, για κάθε service και για κάθε profile, ένα αντίστοιχο πρότυπο, όπου ορίζεται η συγκεκριμένη συμπεριφορά που προβλέπεται για το καθένα, αλλά και επιπλέον χαρακτηριστικά και λειτουργίες που πρέπει να υποστηρίζονται από τις συμβατές συσκευές. Για παράδειγμα, το Glucose Profile (GLP) περιέχει το Glucose Service (GLS), το οποίο περιέχει ένα control-point χαρακτηριστικό, μέσω του οποίου ο Collector μπορεί να εκτελεί λειτουργίες στον Glucose Sensor. Μέσω της XML, ορίζονται οι διάφορες τιμές του χαρακτηριστικού αυτού και οι εντολές στις οποίες αντιστοιχούν. Η συγκεκριμένη συμπεριφορά, όμως, κάθε ρόλου, κατά την εκτέλεσή τους, ορίζεται στο αντίστοιχο πρότυπο.

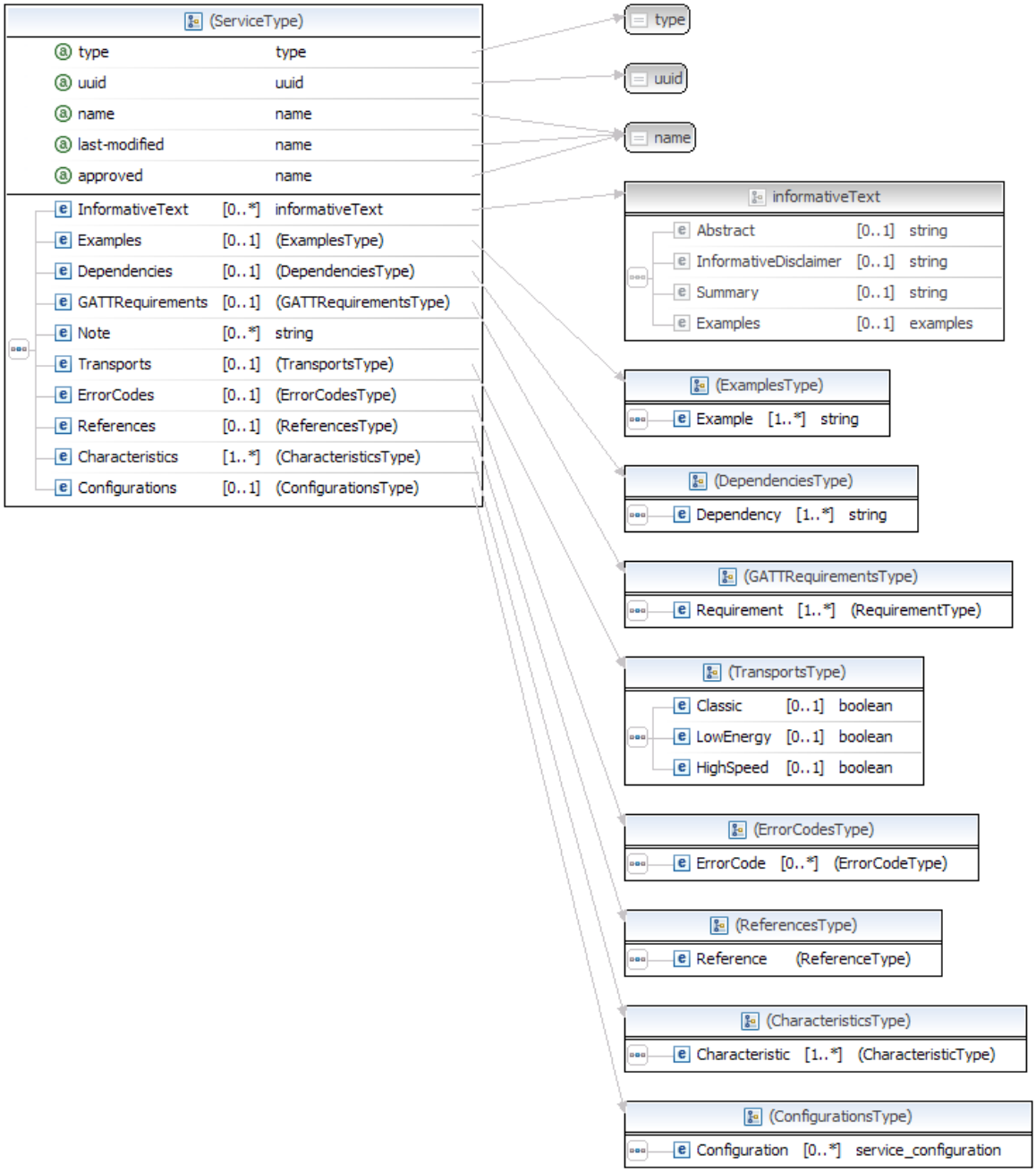
Ακολουθούν διαγράμματα, όπου φαίνονται τα σημαντικότερα XML elements και attributes που χρησιμοποιούνται για τον τυπικό ορισμό των αντικειμένων του GATT.



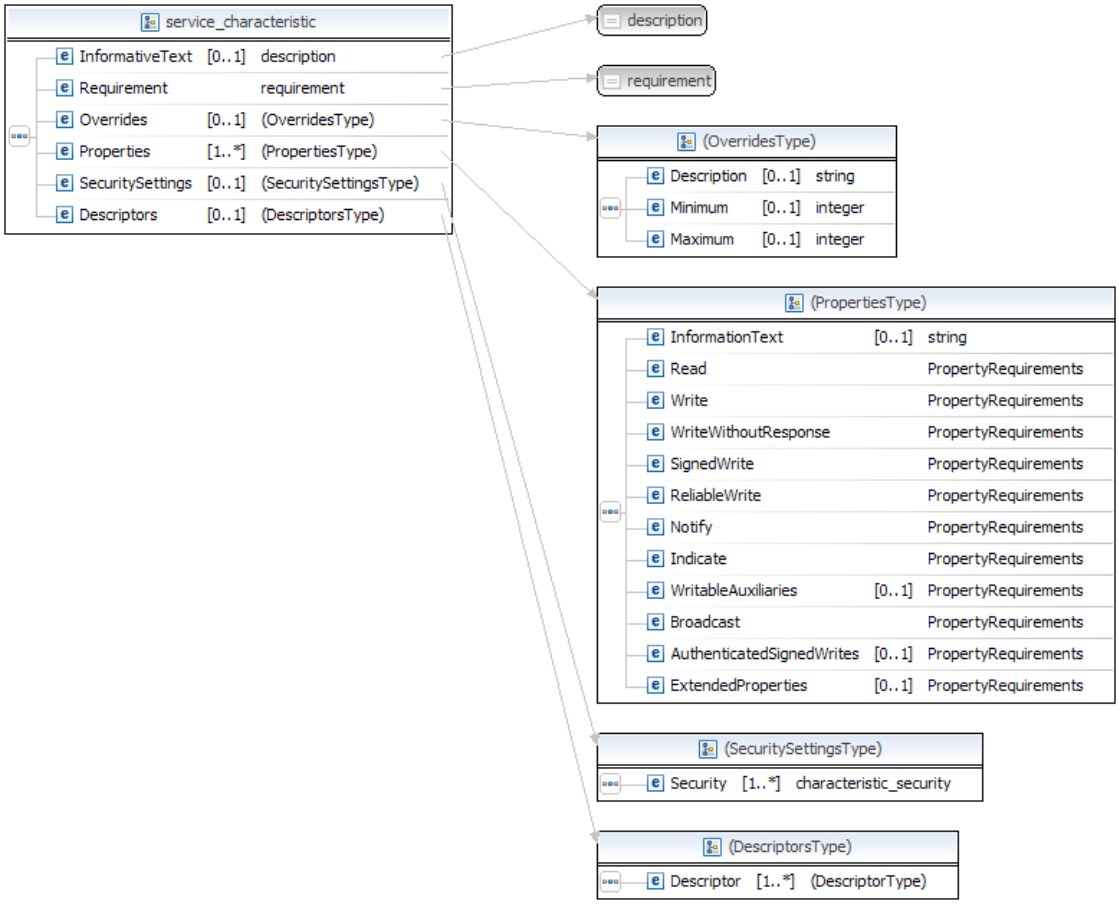
Ορισμός χαρακτηριστικού.



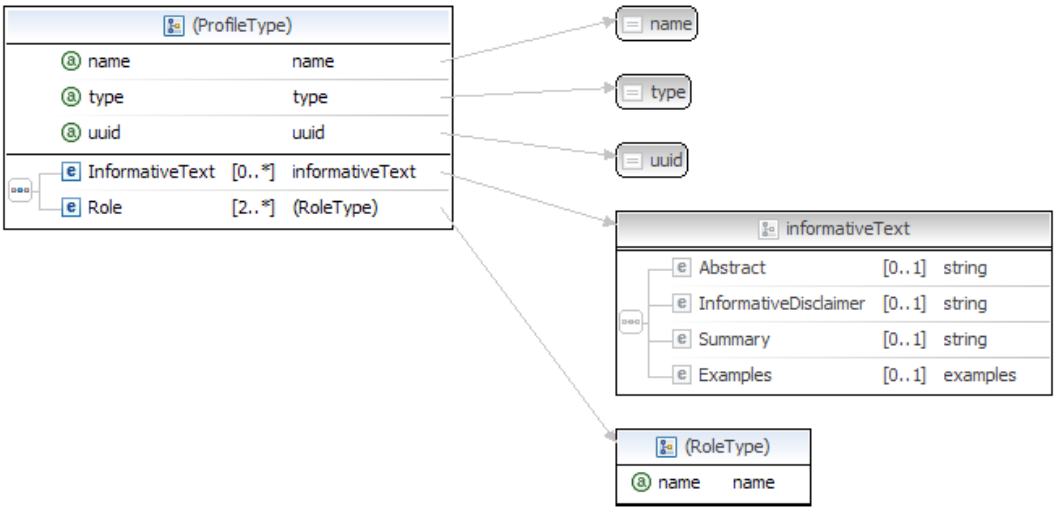
Ορισμός πεδίου τιμής ενός χαρακτηριστικού.



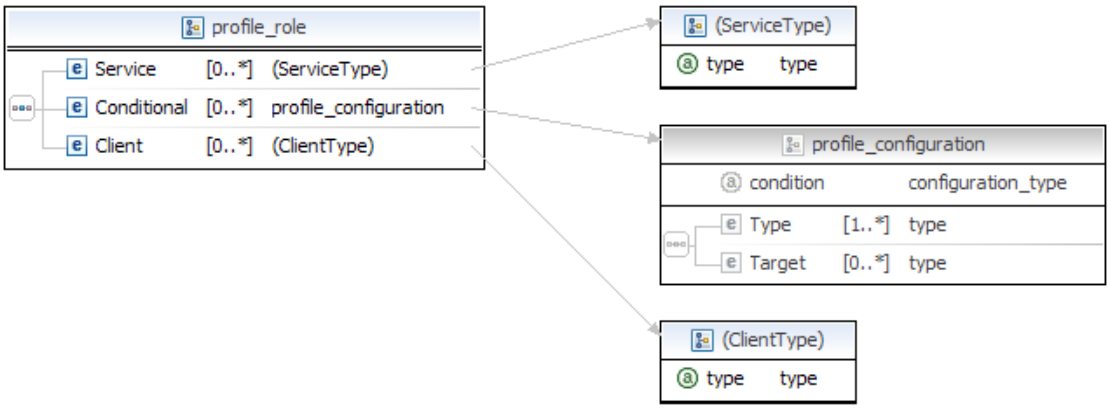
Ορισμός υπηρεσίας.



Ορισμός χαρακτηριστικού υπηρεσίας.



Ορισμός προφίλ.



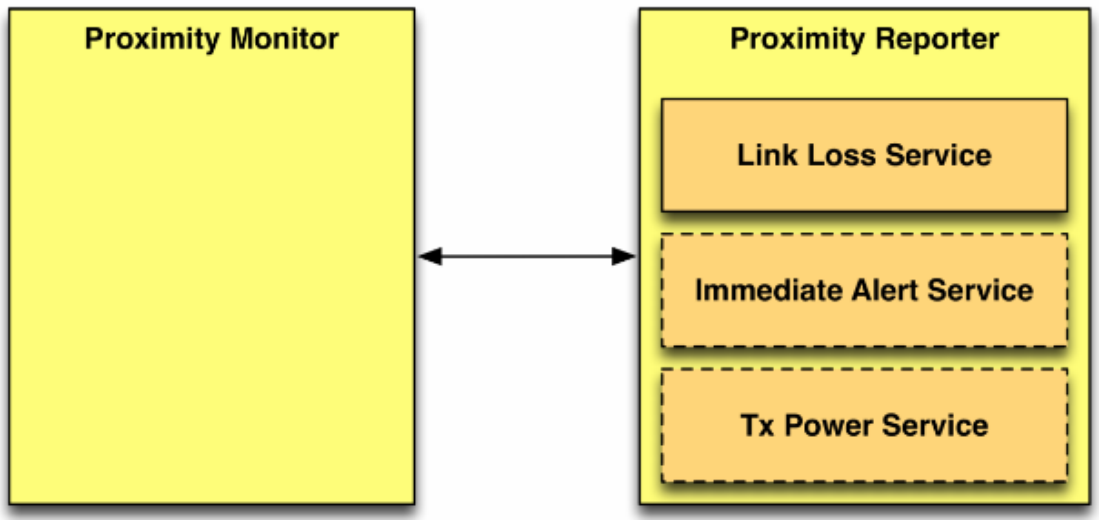
Ορισμός ρόλου προφίλ.

Ένας developer BLE εφαρμογής για την οποία δεν υπάρχουν έτοιμα προφίλ, πρέπει να δημιουργήσει ένα δικό του προφίλ, και, αντίστοιχα, υπηρεσίες και χαρακτηριστικά, που αφορούν την περίπτωση χρήσης που πρόκειται να υλοποιήσει. Όπου είναι δυνατόν, είναι καλύτερο να χρησιμοποιούνται τα παρεχόμενα από το SIG αντικείμενα GATT, ώστε να αποφεύγεται η χρήση custom 128-bit UUIDs, τα οποία είναι λιγότερα αποδοτικά και έχουν μεγαλύτερο κόστος επικοινωνίας.

3.4.2 Proximity Profile

Σε αυτή την υπο-ενότητα περιγράφουμε το Proximity Profile, ένα adopted profile του SIG, για εφαρμογές ελέγχου εγγύτητας, ώστε να δούμε πώς τα παραπάνω χρησιμοποιούνται στην πράξη.

Το Proximity Profile ορίζει δύο ρόλους: Proximity Monitor και Proximity Reporter. Το Proximity Monitor είναι ένας GATT client, ενώ ο Proximity Reporter είναι GATT server. Ο Reporter πρέπει να υποστηρίζει υποχρεωτικά την υπηρεσία Link Loss Service (LLS) και, προαιρετικά, αλλά όχι μόνο τη μία, τις Immediate Alert Service (IAS) και Tx Power Service (TPS).



Το LLS ορίζει τη συμπεριφορά των συσκευών σε περίπτωση που η σύνδεσή τους διακοπεί. Χρησιμοποιεί το χαρακτηριστικό Alert Level, το οποίο καθορίζει το είδος της ειδοποίησης (No – Mild – High Alert) προς τον χρήστη. Το ίδιο χαρακτηριστικό χρησιμοποιείται και από το IAS. Η διαφορά είναι ότι, στο μεν LLS, το Alert Level ορίζει το είδος της ειδοποίησης, όταν διακοπεί η σύνδεση μεταξύ Monitor και Reporter, στο δε IAS, χρησιμοποιείται από το Monitor προκειμένου να ξεκινήσει ή να σταματήσει το alert του Reporter. Παρατηρούμε σε αυτό το σημείο, πώς, το ίδιο χαρακτηριστικό, αποκτά διαφορετική συμπεριφορά σε διαφορετικά services. Το TPS δίνει τη δυνατότητα στο Monitor να ανακτήσει την ισχύ του σήματος εκπομπής του Reporter, η οποία, σε συνδυασμό με το RSSI, που μετρείται τοπικά, μπορεί να δώσει κατά προσέγγιση την απόσταση των δύο συσκευών.

Το Proximity Profile χρησιμοποιεί το LLS για την περίπτωση χρήσης, στην οποία ο χρήστης ειδοποιείται, όταν βγει εκτός μιας περιοχής γύρω από τη συσκευή Reporter, οπότε η σύνδεση μεταξύ τους διακόπτεται. Επίσης συνδυάζει τις άλλες δύο υπηρεσίες, προκειμένου να υλοποιήσει την περίπτωση χρήσης, κατά την οποία ο χρήστης ειδοποιείται, όχι όταν χαθεί η σύνδεση με το Reporter, αλλά όταν η απόσταση από αυτόν ξεπεράσει ένα συγκεκριμένο όριο. Σε αυτή την περίπτωση, όπως προβλέπει το αντίστοιχο πρότυπο, το Monitor γράφει την τιμή του Alert Level στο IAS, εκκινώντας την ειδοποίηση από τον Reporter, στο είδος που ορίζει το Monitor. Αν η απόσταση πέσει κάτω από το όριο, τότε το Monitor ξαναγράφει την τιμή του Alert Level για να σταματήσει την ειδοποίηση. Παρατηρούμε εδώ, πώς ένα profile συνδυάζει υπηρεσίες που παρέχουν συγκεκριμένες συμπεριφορές, προκειμένου να υλοποιήσει μια νέα λειτουργία.

Αξίζει εδώ να αναφερθεί ότι το IAS χρησιμοποιείται και στο Find Me Profile, όπου ορίζονται οι ρόλοι των Locator (GATT client) και Target (GATT server). Σε αυτή την περίπτωση, ο Locator μπορεί να εκκινήσει την ειδοποίηση του Target μέσω του IAS, ώστε ο χρήστης να μπορεί να εντοπίσει το αντικείμενο που θέλει. Η εκκίνηση αυτή, όμως, σε αντίθεση με το Proximity Profile, δε γίνεται αυτόματα, αλλά μετά από ενέργεια του χρήστη. Παρατηρούμε εδώ, πώς το ίδιο service, και η συμπεριφορά που ορίζει, μπορεί να χρησιμοποιείται με άλλους τρόπους από διαφορετικά προφίλ, ώστε να υλοποιούνται διαφορετικές περιπτώσεις χρήσης.

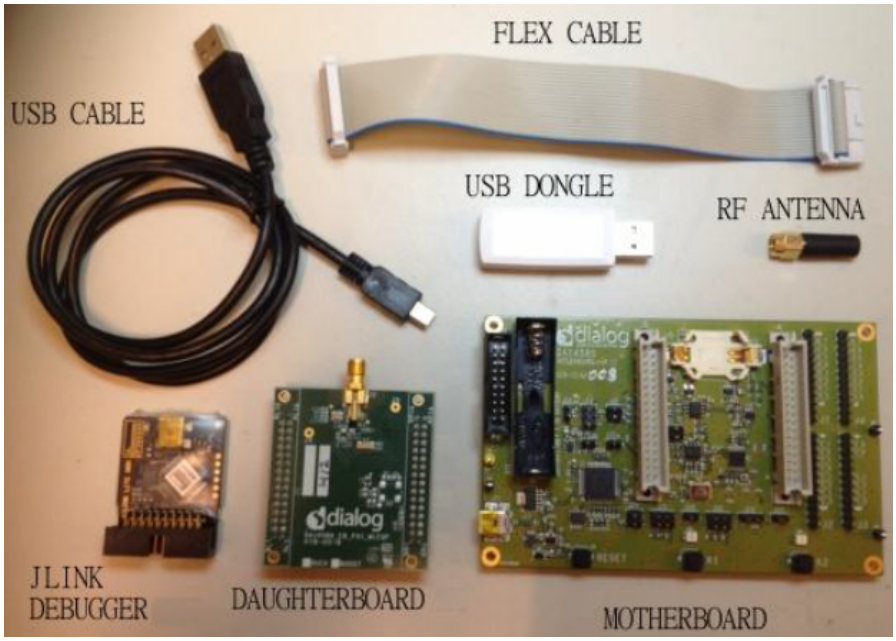
Με βάση τα παραπάνω, μπορούμε εύκολα να διαπιστώσουμε ότι, αν μια συσκευή περιέχει το IAS, τότε είναι δυνατόν σε μια άλλη συσκευή να εκκινήσει την ειδοποίηση από την πρώτη, άσχετα αν αυτές υλοποιούν ή όχι τα αντίστοιχα προφίλ. Εφόσον το IAS ορίζει μια συγκεκριμένη συμπεριφορά, αυτή αναμένεται να είναι η ίδια σε όποιο προφίλ και αν χρησιμοποιείται. Έτσι, μια συσκευή που αναγνωρίζει το IAS, αλλά όχι τις υπόλοιπες υπηρεσίες ενός GATT server, μπορεί, αν θέλει, να το χρησιμοποιήσει για τη δική της λειτουργικότητα. Εδώ φαίνεται ένα από τα χαρακτηριστικά του μοντέλου δεδομένων που ορίζεται με τη βοήθεια του GATT. Το μοντέλο αυτό είναι βασισμένο σε υπηρεσίες, οι οποίες συνδυάζονται μεν σε περιπτώσεις χρήσης μέσω των προφίλ, αλλά υπάρχουν και ανεξάρτητα ως συγκεκριμένες συμπεριφορές. Έτσι, ένας client, παρόλο που δεν είναι συμβατός με το προφίλ ενός server, μπορεί να χρησιμοποιήσει τις υπηρεσίες του τελευταίου, τις οποίες αναγνωρίζει. Για παράδειγμα, μία συσκευή μπορεί να χρησιμοποιεί ένα θερμόμετρο για να πραγματοποιήσει μια εξειδικευμένη λειτουργία. Η λειτουργία αυτή ορίζεται στο αντίστοιχο προφίλ. Όμως, η συσκευή μπορεί, ανάμεσα στα custom services του profile, να παρέχει μια γνωστή υπηρεσία μετρήσεων θερμοκρασίας. Ένας μη συμβατός client, παρόλο που δεν μπορεί να χρησιμοποιήσει όλη τη λειτουργικότητα της συσκευής, μπορεί παρόλα αυτά, αν αναγνωρίζει την υπηρεσία μετρήσεων θερμοκρασίας, να ανακτά μετρήσεις θερμοκρασίας από αυτήν.

Κεφάλαιο 4

DA14580 Development Board

Για την υλοποίηση του τμήματος του συστήματος καταγραφής καιρού, το οποίο, μέσω μιας σειράς κατάλληλων αισθητήρων, παρέχει δυνατότητα μετρήσεων δεδομένων σχετικών με τον καιρό, χρησιμοποιήσαμε το ενσωματωμένο σύστημα DA14580.

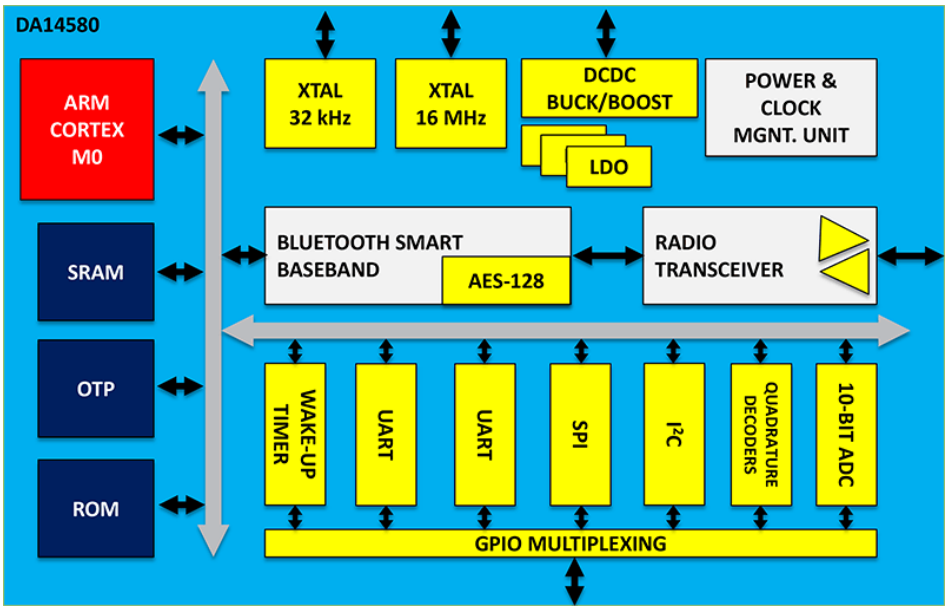
Το DA14580 είναι ένα Bluetooth Low Energy System on a Chip (SoC), το οποίο παράγεται από την εταιρία Dialog Semiconductor. Το SoC αυτό περιλαμβάνει όλα τα απαραίτητα στοιχεία για την υλοποίηση BLE εφαρμογών, όπως micro-controller, μνήμη, ROM για την αποθήκευση του προγράμματος, καθώς και μια σειρά από περιφερειακά και GPIOs για τη σύνδεση του με εξωτερικά στοιχεία της εφαρμογής. Παράλληλα, περιλαμβάνει BLE controller και φυσικό επίπεδο (baseband), ενώ το firmware του υλοποιεί όλη τη στοίβα πρωτοκόλλων του BLE, παρέχοντας, επιπλέον, μια σειρά από υπηρεσίες και λειτουργίες, πάνω στις οποίες μπορούν να χτιστούν οι εφαρμογές. Το DA14580 είναι ένα small form factor chipset (η μικρότερη εκδοχή του είναι μόλις 2.5 x 2.5 x 0.5 mm), με το οποίο είναι δυνατόν, με χρήση ελάχιστων εξωτερικών στοιχείων, να δημιουργηθεί ένα πλήρες Bluetooth Smart σύστημα. Παράλληλα, εξαιτίας της μικρής κατανάλωσης κατά την αποστολή και λήψη δεδομένων, και της σχεδόν μηδενικής κατανάλωσης, όταν περνάει σε sleep mode, μπορεί να λειτουργεί με μπαταρία για μεγάλα χρονικά διαστήματα. Τα παραπάνω το καθιστούν ιδανικό για μια πλειάδα εφαρμογών BLE, στις οποίες απαιτούνται μικρές σε μέγεθος και κόστος συσκευές, ικανές να λειτουργούν ακόμα και για χρόνια με μία κοινή μπαταρία.



Για το prototyping και debugging BLE εφαρμογών, βασισμένων στο DA14580, η εταιρία Dialog Semiconductor παρέχει μία αναπτυξιακή πλακέτα και μια σειρά από εργαλεία, με σκοπό τον έλεγχο των εφαρμογών, πριν αυτές περάσουν στην παραγωγή. Ταυτόχρονα, παρέχεται σε μορφή κώδικα ένα framework, το οποίο, βασισμένο στις υπηρεσίες που παρέχονται από το firmware, υλοποιεί ένα μεγάλο μέρος των απαραίτητων λειτουργιών του συστήματος, δίνοντας τη δυνατότητα στους developers να επικεντρωθούν περισσότερο στο κομμάτι της εφαρμογής. Αυτό συμπληρώνεται με την υλοποίηση πολλών από τα SIG adopted profiles, τα οποία προσφέρονται έτοιμα προς ενεργοποίηση και χρήση.

Σε αυτό το κεφάλαιο αναφέρουμε συνοπτικά τα δομικά στοιχεία, από τα οποία αποτελείται το DA14580 και, στη συνέχεια, περιγράφουμε τον τρόπο λειτουργίας του συστήματος και τις υπηρεσίες και λειτουργίες που αυτό παρέχει, τις οποίες χρησιμοποιήσαμε για την υλοποίηση της ενσωματωμένης εφαρμογής.

4.1 Στοιχεία του συστήματος



Στην παραπάνω εικόνα παρουσιάζονται τα βασικά στοιχεία, από τα οποία αποτελείται το DA14580 SoC. Το Bluetooth Smart Baseband περιέχει εξειδικευμένο επεξεργαστή για το Link Layer, ενώ περιλαμβάνει και hardware υλοποίηση του αλγόριθμου κρυπτογράφησης AES-128. Στη ROM των 84 KB, περιέχεται το firmware, το οποίο περιέχει την υλοποίηση του BLE Protocol Stack με όλα τα απαραίτητα στοιχεία (L2CAP, SMP, ATT, GATT, GAP). Παράλληλα, περιέχει ένα μικρό real-time operating system, ο πυρήνας του οποίου ορίζει το πλαίσιο, μέσα στο οποίο τρέχουν όλα τα υπόλοιπα στοιχεία του συστήματος. Το firmware αυτό αναπτύχθηκε από την εταιρία RivieraWaves, από την οποία η Dialog Semiconductor απέκτησε τα δικαιώματα χρήσης.

4.1.1 Επεξεργαστής

Για την εκτέλεση του firmware και του κώδικα των εφαρμογών, το DA14580 περιλαμβάνει τον μικροελεγκτή ARM Cortex-M0, ο οποίος είναι ένας 32-bit RISC επεξεργαστής, χρονισμένος στα 16 MHz. Ο Cortex-M0 αναπτύχθηκε από την εταιρία ARM Holdings. Σε αντίθεση με άλλες εταιρίες κατασκευής επεξεργαστών, η ARM Holdings δεν

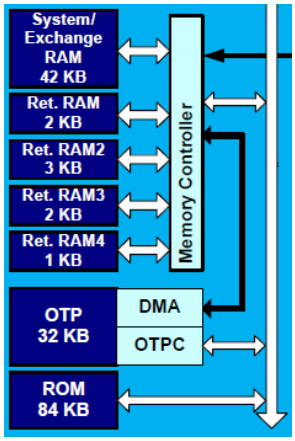
παράγει η ίδια τους επεξεργαστές που αναπτύσσει. Αυτό που κάνει, είναι να δίνει τα δικαιώματα χρήσης τους σε άλλες κατασκευάστριες εταιρίες, παρέχοντας τα σχέδια των επεξεργαστών σε κατάλληλη hardware description language (HDL). Αυτό δίνει τη δυνατότητα στους κατασκευαστές να ενσωματώσουν τον επεξεργαστή στα δικό τους σχεδιασμό hardware, απενεργοποιώντας, ενδεχομένως, στοιχεία που δεν τους είναι απαραίτητα.

Ο Cortex-M0 ανήκει στη σειρά Cortex-M της ARM, η οποία περιέχει πυρήνες 32-bit, που προορίζονται για χρήση ως μικροελεγκτές σε ενσωματωμένα συστήματα. Τρέχει το επονομαζόμενο Thumb instruction set, με ορισμένες επιπλέον εντολές από το Thumb-2. Το Thumb instruction set είναι μια παραλλαγή του 32-bit instruction set των ARM επεξεργαστών, το οποίο περιλαμβάνει απλοποιημένες εντολές των 16-bit. Αυτό είναι αρκετά σημαντικό για τα ενσωματωμένα συστήματα, αφού η πυκνότητα του κώδικα είναι μεγαλύτερη και, έτσι, απαιτείται λιγότερη μνήμη για την αποθήκευση και εκτέλεση των προγραμμάτων. Σε αντίθεση με άλλους ARM επεξεργαστές, ο Cortex-M0 λειτουργεί πάντα σε Thumb mode. Παρότι περιλαμβάνει μόνο 56 εντολές, αυτές είναι βελτιστοποιημένες, ώστε να μπορούν να εκτελούν τις συνηθισμένες λειτουργίες των προγραμμάτων.

Ο Cortex-M0 περιλαμβάνει ένα sleep mode, στο οποίο παρουσιάζει ελάχιστη κατανάλωση. Υποστηρίζει δυνατότητες debugging χρησιμοποιώντας το πρωτόκολλο Serial Wire Debug (SWD), το οποίο είναι παρόμοιο με το JTAG, με το πλεονέκτημα ότι απαιτεί μόνο δύο καλώδια. Παρέχει επίσης δυνατότητες programmable interrupts με επίπεδα προτεραιότητας, μέσω του Nested Vectored Interrupt Controller (NVIC), 24-bit System Tick timer και SuperVisor Call (SVC) για την υποστήριξη λειτουργικών συστημάτων, καθώς και Wake-Up Interrupt Controller (WIC), ο οποίος δίνει τη δυνατότητα απενεργοποίησης του επεξεργαστή όταν είναι σε sleep mode, επιτρέποντας όμως σε εξωτερικές διακοπές να τον ενεργοποιήσουν. Ένα άλλο χαρακτηριστικό των ARM επεξεργαστών, που είναι σημαντικό για ενσωματωμένα και real-time συστήματα, είναι η ικανότητα τους να έχουν αναμενόμενο και σταθερό latency στην εκτέλεση των interrupt handlers.

4.1.2 Μνήμες

Εκτός της ROM, η οποία, όπως αναφέραμε παραπάνω, περιέχει το firmware του συστήματος, υπάρχουν ακόμα η OTP μνήμη, η System RAM, και οι Retention RAMs.



- **OTP (One Time Programmable):** 32 KB μνήμη για την αποθήκευση του προγράμματος, καθώς και πληροφοριών και παραμέτρων της συσκευής, όπως το Bluetooth address. Όπως δηλώνει και το όνομά της, μπορεί να γραφτεί μόνο μία φορά. Μετά την εκκίνηση του συστήματος, όπως και μετά από Deep Sleep Mode, τα περιεχόμενά της μεταφέρονται στη System RAM, από όπου και εκτελούνται.

- **System RAM**: 42 KB μνήμη, όπου τοποθετείται το πρόγραμμα προκειμένου να εκτελεστεί. Σε αυτήν αποθηκεύονται επίσης δεδομένα, τα οποία δεν είναι απαραίτητα μετά από Deep Sleep Mode, ή μπορούν να ξαναδημιουργηθούν.

- **Retention RAMs**: Τέσσερις μνήμες χωρητικότητας 2KB, 3KB, 2KB και 1KB. Οι συγκεκριμένες μνήμες έχουν πολύ μικρή κατανάλωση ενέργειας και χρησιμοποιούνται για τη διατήρηση των απαραίτητων δεδομένων του προγράμματος και του BLE stack, κατά τη διάρκεια του Deep Sleep Mode. Κάθε μία από αυτές

μπορεί να ενεργοποιηθεί ανεξάρτητα από τις υπόλοιπες, δίνοντας στην εφαρμογή επιπλέον έλεγχο της κατανάλωσης σε Deep Sleep Mode.

4.1.3 Περιφερειακά

Το DA1580 περιλαμβάνει μια σειρά από περιφερειακά για τη σύνδεση με εξωτερικά στοιχεία. Ο έλεγχος των περιφερειακών γίνεται μέσω ενός συνόλου από καταχωρητές που παρέχονται από το σύστημα, οι οποίοι μπορούν να χρησιμοποιηθούν για να ελέγξουν όλες τις λειτουργίες τους. Για τα περισσότερα περιφερειακά παρέχονται επίσης drivers, είτε από το firmware, είτε σε μορφή κώδικα στο SDK του αναπτυξιακού, καθιστώντας έτσι ευκολότερη τη χρήση τους. Για τη χρήση των περιφερειακών του συστήματος, παρέχεται ένας αριθμός από General-Purpose Input/Output (GPIO) pins, τα οποία μπορούν να οριστούν προγραμματιστικά ως είσοδοι ή έξοδοι, είτε γενικής φύσης, είτε κάποιου συγκεκριμένου περιφερειακού. Τα GPIO pins έχουν τη δυνατότητα να διατηρούν την τιμή τους κατά τη διάρκεια του sleep mode. Ο αριθμός των GPIO pins αποτελεί και τη διαφορά μεταξύ των διαθέσιμων form-factors του DA1580.

Τα περιφερειακά που περιλαμβάνονται είναι τα ακόλουθα:

- **UART x 2**: Σειριακές θύρες.
- **SPI (Serial Peripheral Interface)**: Διάδρομος σύνδεσης εξωτερικών περιφερειακών.
- **I^2C** : Διάδρομος σύνδεσης εξωτερικών περιφερειακών.
- **ADC (Analog-to-Digital Converter)**: 10-bit ADC για μέτρηση τάσης (single-ended or differential). Επιτρέπει επίσης τη μέτρηση του επιπέδου της μπαταρίας.
- **Quadrature Decoder**: Χρησιμοποιείται για την αποκωδικοποίηση των σημάτων από έναν rotary encoder, όπως αυτοί που περιλαμβάνονται σε συσκευές τύπου HID (π.χ. mouse), και στους τρεις αξονες (X, Y, Z).
- **Keyboard Controller**: Χρησιμοποιείται για τη σύνδεση ενός keyboard στα GPIO pins, παρέχοντας δυνατότητα debouncing.
- **Timers**: 2 Software Timers, με προγραμματιζόμενη συχνότητα και duty cycle και δυνατότητα ορισμού διακοπών. Wake-Up Timer, που παρέχει τη δυνατότητα εξόδου από sleep mode, όταν ο αριθμός των εξωτερικών διακοπών ξεπεράσει κάποιο προγραμματιζόμενο όριο. Watchdog Timer, που μπορεί να παράγει διακοπή Non-Maskable Interrupt (NMI) ή να προκαλέσει system reset σε περίπτωση που το software παρουσιάσει κάποιο πρόβλημα (π.χ. endless loop).

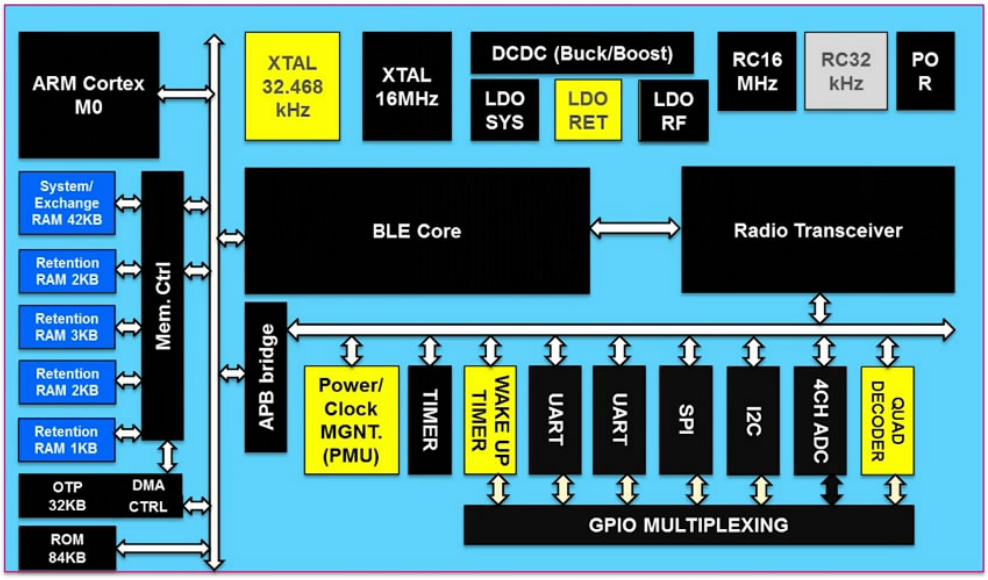
4.2 Sleep Modes

Εκτός από το Active Mode, κατά το οποίο το σύστημα είναι ενεργό και λειτουργεί σε πλήρη ταχύτητα, το DA1580 ορίζει τρία sleep modes. Αυτά, με αυξανόμενο τον αριθμό των απενεργοποιημένων στοιχείων και μειούμενη την κατανάλωση ενέργειας, είναι τα ακόλουθα:

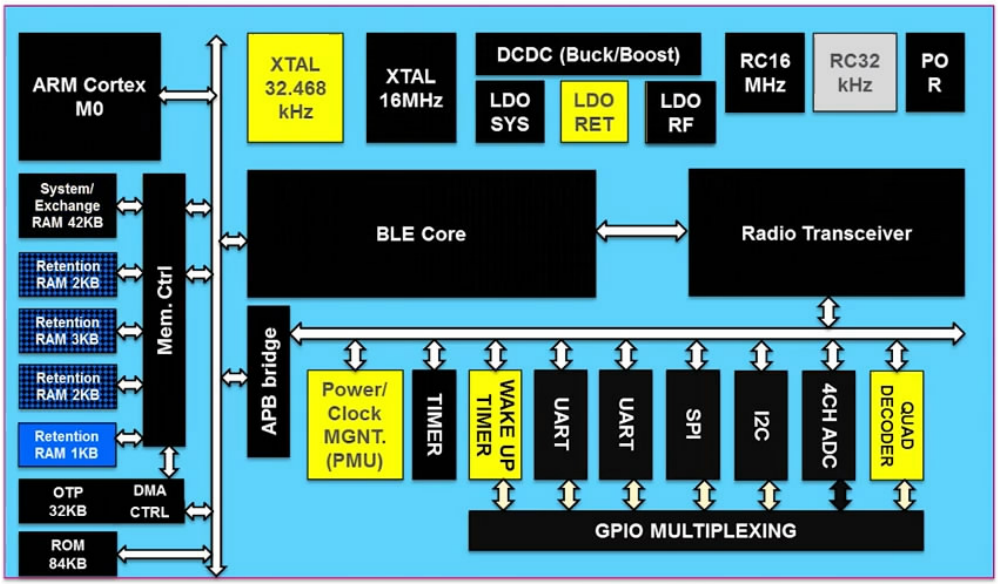
- **Sleep Mode**: Είναι το συνηθισμένο sleep mode στο οποίο μπαίνει ο επεξεργαστής, όταν όλες οι λειτουργίες του προγράμματος έχουν ολοκληρωθεί. Ο επεξεργαστής είναι σε κατάσταση idle και αναμένει κάποιο interrupt. Τα power domains του συστήματος είναι ενεργά, ενώ για τα περιφερειακά ισχύει ότι έχει προγραμματιστεί στους καταχωρητές του συστήματος.

- **Extended Sleep Mode**: Τα περισσότερα power domains του συστήματος είναι απενεργοποιημένα. Ο BLE controller, το Radio Tranceiver καθώς και όλα τα

περιφερειακά είναι κλειστά. Παραμένουν ανοιχτές η System RAM και οι Retention RAMs, καθώς και τα στοιχεία του συστήματος που μπορούν να το ενεργοποιήσουν ξανά, όπως το Power Management Unit (PMU), ο Wake-Up Timer και ο Quadrature Decoder. Η κατανάλωση ενέργειας είναι πολύ μικρή. Εφόσον η System RAM παραμένει ανοιχτή, δε χρειάζεται να μεταφερθεί ξανά το πρόγραμμα από την OTP στη μνήμη (OTP mirroring), μετά την έξοδο από το sleep mode.



- **Deep Sleep Mode**: Εκτός από όλα τα υπόλοιπα στοιχεία που είναι απενεργοποιημένα κατά το Extended Sleep Mode, στο Deep Sleep Mode κλείνει και η System RAM, ενώ είναι δυνατόν να απενεργοποιηθούν, με κατάλληλες ρυθμίσεις σε καταχωρητές του συστήματος, και κάποιες από τις Retention RAMs. Παραμένουν ανοιχτές μόνο όσες Retention RAMs έχουν ρυθμιστεί κατά αυτόν τον τρόπο, και, όπως και πριν, τα στοιχεία του συστήματος που μπορούν να το ενεργοποιήσουν ξανά. Η κατανάλωση ενέργειας είναι η ελάχιστη δυνατή. Εφόσον η System RAM απενεργοποιείται, πρέπει το πρόγραμμα να μεταφερθεί από την OTP στη System RAM, αμέσως μετά την έξοδο από το sleep mode.



Το application framework του DA14580 SDK δίνει πλήρη έλεγχο στους developers πάνω στις διαδικασίες που έχουν να κάνουν με τα sleep modes. Τα Extended και Deep Sleep Mode μπορούν να απενεργοποιηθούν εντελώς, κατά το configuration της εφαρμογής, ή να ενεργοποιηθεί μόνο το Extended και όχι το Deep Sleep Mode. Επίσης, ανεξάρτητα από την ενεργοποίηση των διαφόρων sleep modes, η εφαρμογή έχει πλήρη έλεγχο της όλης διαδικασίας. Μπορεί, δηλαδή, ανάλογα με την τρέχουσα κατάσταση στην οποία βρίσκεται, να αποφασίσει ότι το σύστημα δεν πρέπει να περάσει σε sleep mode. Για την ακρίβεια, η εφαρμογή μπορεί να εκκινήσει ένα sleep mode κατά την εκτέλεση της, ακόμα και αν αυτό δεν είχε ενεργοποιηθεί κατά το configuration. Αυτό, βέβαια, δεν ενδείκνυται, γιατί, σε κάθε sleep mode, και ειδικότερα στο Deep, όπου η System Ram κλείνει, πρέπει να χρησιμοποιείται το αντίστοιχο memory map, ώστε να διατηρούνται στις Retention RAMs όλα τα δεδομένα που είναι απαραίτητα για τη λειτουργία του συστήματος. Αυτά περιλαμβάνουν τόσο δεδομένα της εφαρμογής, όσο και δεδομένα του πυρήνα του firmware, και, βέβαια, τα δεδομένα του BLE stack, όπως ουρές λήψης και αποστολής.

Ο developer, λοιπόν, ορίζει το επιθυμητό sleep mode, ενεργοποιώντας τα κατάλληλα preprocessor macros κατά το compilation της εφαρμογής. Επίσης, φροντίζει να δηλώσει τα δεδομένα του προγράμματος του, που είναι απαραίτητο να διατηρούνται μετά από Deep Sleep Mode, με τέτοιο τρόπο, ώστε αυτά να τοποθετηθούν σε κάποια από τις Retention RAMs. Επειδή, όμως, ο χώρος σε αυτές είναι περιορισμένος, και, ανάλογα με τις ρυθμίσεις του συστήματος, μεγάλο μέρος αυτών μπορεί να χρησιμοποιείται από το ίδιο το σύστημα, μπορεί να χρειαστεί ειδική μέριμνα κατά τον ορισμό του memory map, ώστε όλα τα απαραίτητα δεδομένα να χωράνε στις Retention RAMs.

Ένα άλλο ζήτημα που πρέπει να ληφθεί υπόψη κατά την επιλογή του sleep mode, είναι και το γεγονός ότι μετά από κάθε Deep Sleep Mode, πρέπει να ακολουθεί OTP mirroring, το οποίο, βέβαια, απαιτεί κάποια ενέργεια προκειμένου να εκτελεστεί. Υπάρχει δηλαδή κάποιο όριο διάρκειας του Deep Sleep Mode, κάτω από το οποίο, το όφελος από την είσοδο σε Deep Sleep Mode, αναιρείται από την απαιτούμενη ενέργεια για το OTP mirroring.

4.3 Τρόπος λειτουργίας του πυρήνα

Μέσα στο firmware του DA14580, περιέχεται ένα, μικρού μεγέθους και αποδοτικό, real-time operating system, ο πυρήνας (kernel), το οποίο ορίζει τα βασικά δομικά στοιχεία και παρέχει κατάλληλες υπηρεσίες και λειτουργίες, πάνω στις οποίες χτίζονται όλα τα υπόλοιπα στοιχεία του συστήματος και οι εφαρμογές.

Ο kernel παρέχει τις ακόλουθες δυνατότητες:

- Ανταλλαγή μηνυμάτων (messages)
- Σώσιμο μηνυμάτων για αποστολή σε επόμενη χρονική στιγμή.
- Ορισμός timers
- Ορισμός events
- Memory management (malloc style allocation/deallocation)

Τα βασικά δομικά στοιχεία του συστήματος είναι τα λεγόμενα tasks. Ένα task μπορεί να θεωρηθεί αντίστοιχο της διεργασίας σε ένα κανονικό λειτουργικό σύστημα. Κάθε task έχει ένα τύπο και μπορεί να έχει μηδέν ή περισσότερα instances. Το task ορίζει, επίσης, ένα σύνολο καταστάσεων, στις οποίες μπορεί να βρίσκεται ένα instance. Κάθε instance μπορεί να βρίσκεται σε μία μόνο κατάσταση από το σύνολο καταστάσεων του task, ενώ διαφορετικά instances μπορούν να βρίσκονται σε διαφορετικές καταστάσεις. Ο τύπος του task, σε

συνδυασμό με το instance index του, ορίζουν το task ID, το οποίο χρησιμοποιείται από τον kernel ως αναγνωριστικό του task instance.

Κατά τη δημιουργία ενός task, παρέχεται στον kernel μια δομή, η οποία ονομάζεται task descriptor. Μέσω του task descriptor, το task ορίζει:

- Τον συνολικό αριθμό καταστάσεων, στις οποίες μπορεί να βρίσκεται ένα instance του.
- Τα μηνύματα, τα οποία μπορεί να δεχτεί το task, σε κάθε μία από τις καταστάσεις του, και τους αντίστοιχους handlers.
- Τα μηνύματα, που αφορούν όλες τις καταστάσεις του task, και τους αντίστοιχους handlers.
- Τον μέγιστο αριθμό task instances που μπορούν να υπάρχουν ταυτόχρονα.
- Την τρέχουσα κατάσταση, στην οποία βρίσκεται κάθε task instance.

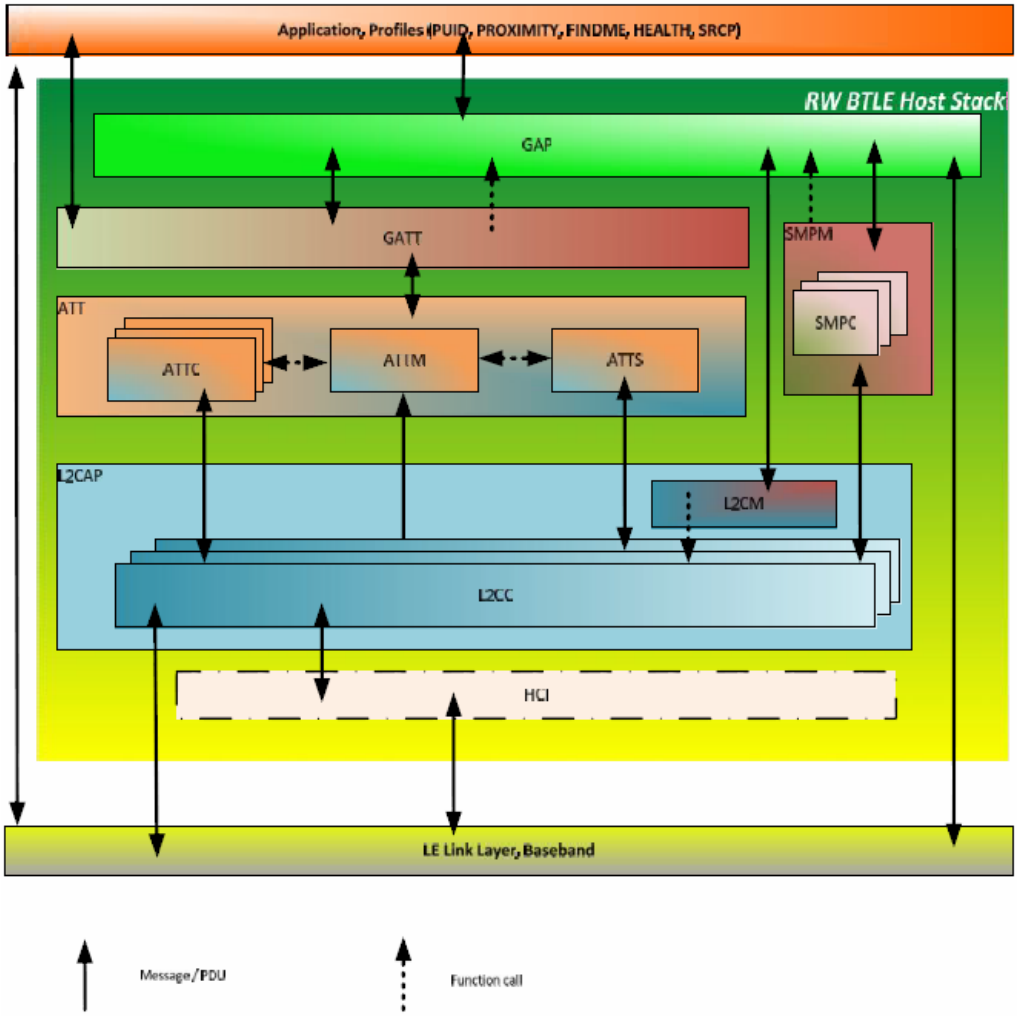
Σύμφωνα με τα παραπάνω, κατά τη λειτουργία του συστήματος, έχουμε ένα σύνολο από tasks και τα αντίστοιχα instances, τα οποία επικοινωνούν μεταξύ τους ανταλλάσσοντας μηνύματα. Ένα μήνυμα μπορεί, για παράδειγμα, να χρησιμοποιηθεί για τη μεταφορά δεδομένων, για ενημέρωση σχετικά με κάποιο γεγονός, ή για την εκτέλεση μιας λειτουργίας ή υπηρεσίας που προσφέρεται από τον παραλήπτη. Κάθε μήνυμα έχει ένα αναγνωριστικό, που καθορίζει τον τύπο του μηνύματος. Περιέχει, επίσης, τα task IDs της πηγής και του προορισμού, καθώς και δεδομένα, το μέγεθος και η δομή των οποίων εξαρτώνται από τον τύπο του μηνύματος. Ένα task μπορεί να ορίσει δικούς του τύπους μηνυμάτων, μέσω των οποίων άλλα tasks μπορούν να επικοινωνούν μαζί του, στέλνοντας ή λαμβάνοντας τα συγκεκριμένα μηνύματα.

Όλα τα modules, που περιέχονται στο firmware, και τα οποία υλοποιούν τη στοίβα πρωτοκόλλων του BLE, υλοποιούνται ως tasks. Καθένα από αυτά ορίζει ένα σύνολο μηνυμάτων, που σχετίζονται με τη λειτουργία του ή τις υπηρεσίες που προσφέρει. Έτσι, ένα BLE stack module μπορεί να ειδοποιεί τα ενδιαφερόμενα tasks για γεγονότα σχετικά με το BLE, ενώ, επίσης, μπορεί να δέχεται αιτήματα για την εκτέλεση διαδικασιών που σχετίζονται με τη λειτουργία του. Τα υλοποιημένα adopted profiles, που περιέχονται στο SDK, ορίζονται επίσης ως tasks. Τέλος, και το ίδιο το application framework, πάνω στο οποίο χτίζεται η εφαρμογή, ορίζει το δικό του task, μέσω του οποίου επικοινωνεί με το BLE stack και με όσα profiles είναι ενεργοποιημένα. Παρατηρούμε, δηλαδή, ότι όλη η λειτουργία του συστήματος βασίζεται στην ανταλλαγή μηνυμάτων.

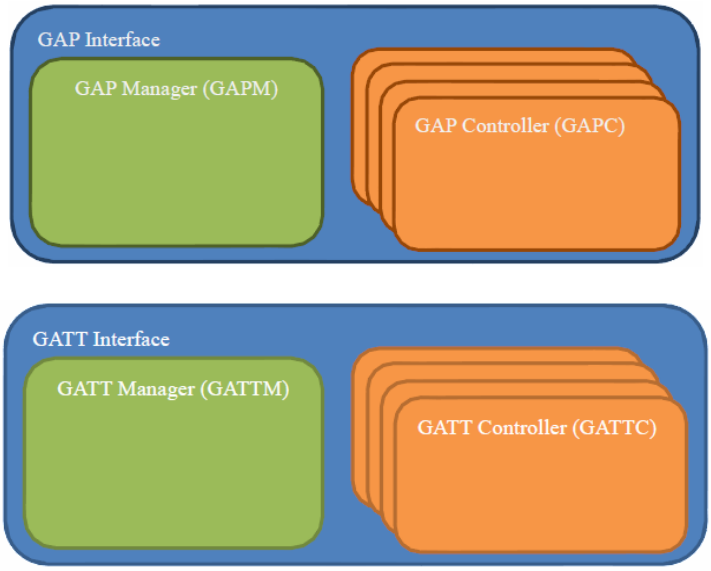
4.4 Αρχιτεκτονική

4.4.1 Γενική Δομή

Στο κατώτερο τμήμα του συστήματος βρίσκεται ο controller, που αποτελείται από το Bluetooth Smart baseband και το Radio Tranceiver. Αυτά, με τη σειρά τους, αποτελούν το επίπεδο ζεύξης δεδομένων και το φυσικό επίπεδο, όπως αυτά ορίζονται στη στοίβα πρωτοκόλλων του BLE. Το baseband περιέχει έναν ξεχωριστό, εξειδικευμένο, επεξεργαστή, ώστε να μη χρειάζεται να χρησιμοποιεί τον κεντρικό επεξεργαστή του συστήματος. Παράλληλα, περιέχει και hardware υλοποίηση του AES-128, για την υποστήριξη των δυνατοτήτων κρυπτογράφησης του πρωτοκόλλου. Πάνω από τον controller βρίσκεται ο host, ο κώδικας του οποίου περιέχεται στο firmware και εκτελείται στον κεντρικό επεξεργαστή. Η επικοινωνία μεταξύ των δύο τμημάτων γίνεται μέσω του Host-Controller Interface (HCI). Στην ακόλουθη εικόνα βλέπουμε τη γενική δομή του host.



Τα στοιχεία των GAP και GATT χωρίζονται περαιτέρω ως εξής:



4.4.2 Managers και Controllers

Το σύστημα, λοιπόν, αποτελείται από ένα σύνολο από managers και controllers. Κάθε manager ή controller έχει τον δικό του τύπο task και μηδέν ή περισσότερα instances. Επίσης, ορίζει τύπους μηνυμάτων για την επικοινωνία με τα tasks άλλων modules και την εφαρμογή. Ένας manager εκτελεί γενικές λειτουργίες του συστήματος και του πρωτοκόλλου BLE, οι οποίες είναι ανεξάρτητες των συνδέσεων, στις οποίες μπορεί να συμμετέχει κάθε στιγμή ο host. Ένας controller εκτελεί λειτουργίες, οι οποίες σχετίζονται με μια συγκεκριμένη σύνδεση του host με μια απομακρυσμένη συσκευή. Κάθε manager task έχει ένα μόνο instance το οποίο χρησιμοποιείται από όλα τα υπόλοιπα tasks. Αντίθετα, τα controller tasks έχουν ένα instance ανά σύνδεση. Τα instances αυτά δημιουργούνται και καταστρέφονται δυναμικά από τον αντίστοιχο manager, κατά τη σύνδεση και αποσύνδεση. Το instance index τους σχετίζεται άμεσα με το connection index, το οποίο χρησιμοποιείται από το σύστημα για τον διαχωρισμό και αναγνώριση των συνδέσεων.

Για παράδειγμα, στο επίπεδο του GAP έχουμε τον GAP manager. Αυτός ελέγχει λειτουργίες του συστήματος σχετικές με το GAP specification του BLE, οι οποίες είναι ανεξάρτητες συνδέσεων, όπως το advertising interval, τα scan response data, τα GAP Device Name και Appearance, η εκκίνηση advertising ή της διαδικασίας σύνδεσης. Αφού δημιουργηθεί μια νέα σύνδεση με μια απομακρυσμένη συσκευή, ο GAP manager δημιουργεί ένα instance του task GAP controller, το οποίο και διαχειρίζεται τις λειτουργίες που είναι άμεσα συσχετισμένες με τη συγκεκριμένη σύνδεση, όπως bonding, αλλαγή παραμέτρων σύνδεσης, αποσύνδεση.

Όλες οι παραπάνω λειτουργίες γίνονται με την αποστολή μηνυμάτων μεταξύ των tasks. Κάθε module ορίζει το interface του ως ένα σύνολο από τύπους μηνυμάτων και αντίστοιχων δεδομένων, τα οποία μπορούν να χρησιμοποιηθούν από άλλα modules, προκειμένου να εκτελέσουν τις λειτουργίες που παρέχονται από αυτό. Επίσης, ορίζονται και τύποι μηνυμάτων, που αντιστοιχούν σε γεγονότα που προκύπτουν από τη λειτουργία του module, πάνω στους οποίους μπορούν να κάνουν register άλλα modules, μέσω του task descriptor τους, ώστε να ενημερώνονται όταν συμβαίνουν τα γεγονότα αυτά.

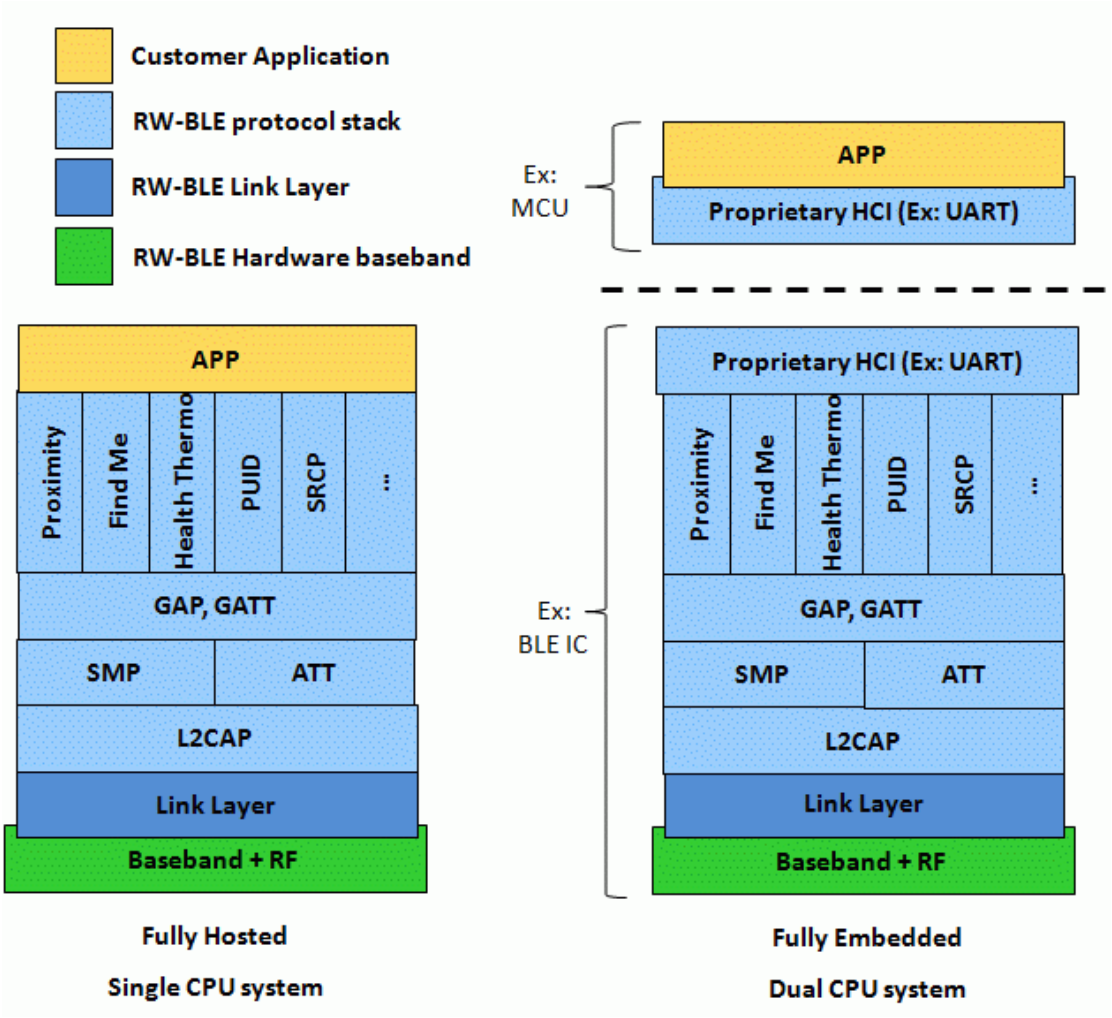
4.4.3 Τύποι αρχιτεκτονικών

Όπως είδαμε κατά την περιγραφή του BLE stack, το HCI ορίζει ένα προτυποποιημένο σύνολο από εντολές και γεγονότα, μέσω των οποίων γίνεται η επικοινωνία μεταξύ host και controller του συστήματος, προκειμένου να εκτελέσουν διαδικασίες του πρωτοκόλλου. Η προτυποποίηση αυτή του HCI δίνει τη δυνατότητα δημιουργίας διαφορετικών αρχιτεκτονικών, ανάλογα με τον αριθμό των διαφορετικών επεξεργαστών που χρησιμοποιούνται για τα εκτελούνται τα συστατικά στοιχεία του συστήματος. Μπορεί, για παράδειγμα, ο controller να βρίσκεται σε διαφορετικό chipset από τον host και η επικοινωνία τους να γίνεται μέσω του προτυποποιημένου HCI.

Το DA14580, εκμεταλλευόμενο το γεγονός ότι όλη η λειτουργία του βασίζεται στην ανταλλαγή μηνυμάτων, επεκτείνει το HCI, ώστε αυτό, εκτός από τις λειτουργίες που ορίζονται στο πρότυπο, να μπορεί να χρησιμοποιηθεί και για την αποστολή και λήψη μηνυμάτων, που ορίζονται από τα διάφορα tasks του συστήματος και την εφαρμογή. Έτσι, δίνει τη δυνατότητα δημιουργίας τύπων αρχιτεκτονικών, όπου ορισμένα tasks του συστήματος μπορεί να τρέχουν σε διαφορετικό επεξεργαστή, επικοινωνώντας με το υπόλοιπο σύστημα, μέσω του ειδικά σχεδιασμένου HCI. Οι βασικοί τύποι αρχιτεκτονικών είναι δύο:

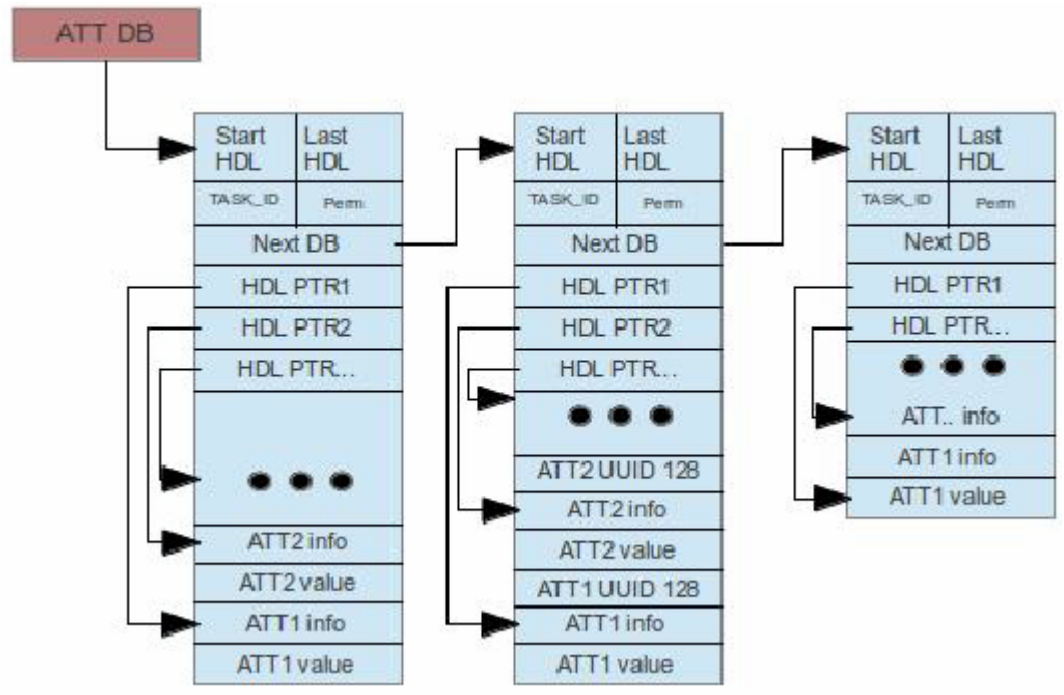
- **Fully Hosted**: Όλα τα στοιχεία του συστήματος, τα ενεργοποιημένα προφίλ, καθώς και η εφαρμογή, τρέχουν στον ίδιο επεξεργαστή (τον κεντρικό επεξεργαστή του συστήματος).

- **Fully Embedded**: Τα στοιχεία του συστήματος και τα ενεργοποιημένα προφίλ τρέχουν στον κεντρικό επεξεργαστή του συστήματος, ενώ η εφαρμογή τρέχει σε εξωτερικό επεξεργαστή, επικοινωνώντας με το υπόλοιπο σύστημα μέσω του proprietary HCI.



4.4.4 Attribute Database

Για την αποθήκευση των attributes του Attribute Protocol , το DA14580 χρησιμοποιεί μία βάση δεδομένων, η οποία αποθηκεύεται στη μνήμη του συστήματος (στις Retention RAMs όταν ενεργοποιείται το Deep Sleep Mode), με τη δομή που φαίνεται στην εικόνα της διπλανής σελίδας:



Έχουμε, δηλαδή, μια λίστα από services, κάθε στοιχείο της οποίας περιέχει τα attributes που περιλαμβάνονται στο service definition της συγκεκριμένης υπηρεσίας. Για κάθε attribute, αποθηκεύονται όλα τα απαραίτητα στοιχεία που ορίζονται στο ATT, όπως handle, type (UUID), permissions και, βέβαια, η τιμή του. Σχετικά με την τιμή, ορίζεται κατά τη δημιουργία του attribute το μέγιστο μέγεθος αυτής, ενώ, στη βάση δεδομένων, αποθηκεύεται επίσης το τρέχον μέγεθος. Υποστηρίζονται τόσο 16-bit SIG UUIDs, όσο και custom 128-bit UUIDs. Οι τιμές των handles των attributes δημιουργούνται ακολουθιακά. Δηλαδή, η τιμή του handle ενός attribute είναι η επόμενη αυτής του προηγούμενου, κάτι που απλοποιεί τον κώδικα ελέγχου των attributes.

Τα permissions ορίζονται πάνω στις βασικές ενέργειες που μπορούν να εκτελεστούν σε κάθε attribute, οι οποίες είναι οι εξής: Read, Write, Notify, Indicate. Για κάθε μία από αυτές μπορεί να καθοριστεί ξεχωριστά αν είναι ενεργοποιημένη, καθώς και το επίπεδο ασφάλειας που απαιτείται, προκειμένου να εκτελεστεί. Έχουμε τρία επίπεδα ασφάλειας:

- Καμία ασφάλεια.
- Απαίτηση unauthenticated κρυπτογραφημένου καναλιού.
- Απαίτηση authenticated κρυπτογραφημένου καναλιού (προστασία από MITM).

Εκτός του απαιτούμενου επίπεδου ασφάλειας, μπορεί να οριστεί επιπλέον η απαίτηση authorization, προκειμένου να εκτελεστεί η αντίστοιχη λειτουργία. Άλλα permissions ενεργοποιούν τη λειτουργία Signed Write, καθώς και την απαίτηση για συγκεκριμένο μήκος κλειδιού κρυπτογράφησης.

Το firmware του DA14580 παρέχει κατάλληλο API για τον χειρισμό της παραπάνω βάσης δεδομένων. Επίσης, κατά τη δημιουργία ενός service, παρέχεται από την εφαρμογή ένα task ID, το οποίο χρησιμοποιείται από το σύστημα, προκειμένου να ειδοποιεί την εφαρμογή για γεγονότα σχετικά με το service, στέλνοντας κατάλληλα μηνύματα στο συγκεκριμένο task. Μέσω αυτών των λειτουργιών επιτυγχάνεται η λειτουργία των προφίλ. Το βασικότερο γεγονός, για το οποίο ενημερώνονται τα tasks των προφίλ, είναι η προσπάθεια εγγραφής κάποιου attribute. Αυτή η εγγραφή σχετίζεται, συνήθως, άμεσα με την περίπτωση χρήσης του συγκεκριμένου προφίλ. Για παράδειγμα, η εγγραφή αυτή μπορεί να αφορά ρυθμίσεις της λειτουργίας και συμπεριφοράς της εφαρμογής, από τον χρήστη της

απομακρυσμένης συσκευής, ή, στην περίπτωση των control-point attributes, εκκίνηση από την εφαρμογή συγκεκριμένων διαδικασιών που προβλέπονται στο προφίλ.

Αξίζει εδώ να σημειωθεί ότι το σύστημα δεν ενημερώνει την εφαρμογή για λειτουργίες ανάκτησης των δεδομένων που περιέχονται στο ATT database. Αν δηλαδή ο client διαβάσει την τιμή ενός attribute, η διαδικασία ολοκληρώνεται στο επίπεδο του ATT και δεν ενημερώνονται τα ανώτερα στρώματα. Το αντίθετο ισχύει για τις εγγραφές. Αν ο client προσπαθήσει να γράψει την τιμή ενός attribute, το task που έχει οριστεί ως υπεύθυνο για το service του attribute, ειδοποιείται και του παρέχονται τα δεδομένα που έστειλε ο client. Μάλιστα, τα δεδομένα αυτά δε γράφονται αυτόματα στη βάση δεδομένων, αλλά μόνο μετά από κλήση της κατάλληλης συνάρτησης του API από την εφαρμογή.

4.5 Application Framework

Κατά την εκκίνηση του συστήματος, ο κώδικας του firmware, αφού εκτελέσει τις βασικές λειτουργίες αρχικοποίησης του συστήματος, περιμένει για 100ms και, στη συνέχεια εκτελεί το OTP mirroring, αντιγράφοντας την εφαρμογή από την OTP στη System RAM. Η αναμονή των 100ms γίνεται, ώστε να εξασφαλιστεί ότι η τάση, που παρέχεται από την μπαταρία, θα έχει σταθεροποιηθεί πριν ξεκινήσει το OTP mirroring. Η παραπάνω διαδικασία δεν εκτελείται στην περίπτωση του debugging, αφού η OTP δε χρησιμοποιείται και είναι κενή, ενώ η εφαρμογή φορτώνεται απευθείας στη μνήμη, μέσω του SWD, και εκτελείται από εκεί.

Μετά ο έλεγχος περνάει στον κώδικα της εφαρμογής με την εκτέλεση του Reset Handler. Ο τελευταίος, αφού επίσης εκτελέσει βασικές λειτουργίες αρχικοποίησης, καλεί τη συνάρτηση main του συστήματος. Η συνάρτηση main περιέχεται στο firmware, και, αφού εκτελέσει τις δικές της αρχικοποιήσεις, καλεί τελικά τη συνάρτηση main_func, η οποία είναι το entry-point της εφαρμογής και παρέχεται σε μορφή κώδικα από το SDK. Για τη σύνδεση του firmware με την εκάστοτε εφαρμογή που εκτελείται στο DA14580, έχουν οριστεί κάποιες δομές δεδομένων, οι οποίες περιέχουν δεδομένα ελέγχου της λειτουργίας του συστήματος και διευθύνσεις συναρτήσεων που καλούνται από το firmware. Οι δομές αυτές τοποθετούνται σε προκαθορισμένες (hardcoded) διευθύνσεις, όπου το firmware μπορεί να τις βρει και να τις χρησιμοποιήσει. Αυτή η τεχνική επιτρέπει στο firmware να μπορεί να λειτουργεί ανεξάρτητα από τις εφαρμογές που τρέχουν στο σύστημα, αφού, οι συναρτήσεις που ορίζουν οι τελευταίες και που καλούνται από το firmware, μπορούν να βρίσκονται οπουδήποτε στη μνήμη, αρκεί οι διευθύνσεις τους να παρέχονται στις αντίστοιχες δομές.

Μια τέτοια δομή είναι το Non Volatile Data Storage (NVDS), μέσω του οποίου ορίζονται δεδομένα του συστήματος όπως advertising και scan response data, το GAP Device name και το Bluetooth address της συσκευής. Άλλη σημαντική δομή είναι το jump table, το οποίο είναι ένας πίνακας, στον οποίο περιλαμβάνονται δεδομένα configuration του συστήματος και διευθύνσεις συναρτήσεων. Σε αυτόν τον πίνακα περιέχεται και η διεύθυνση της main_func. Επίσης, μέσω του jump table, μπορεί να καθοριστεί η αρχιτεκτονική της εφαρμογής, αν δηλαδή είναι fully hosted ή fully embedded.

4.5.1 Main Loop

Η συνάρτηση main_func είναι, όπως είδαμε, το entry-point της εφαρμογής. Αυτή αρχικοποιεί τα υπόλοιπα στοιχεία του συστήματος, καλώντας συναρτήσεις αρχικοποίησης που παρέχονται από το σύστημα και την εφαρμογή, ή θέτοντας κατάλληλες τιμές σε καταχωρητές του συστήματος. Για παράδειγμα, κατά την εκκίνηση της main_func έχουμε

αρχικοποίηση της NVDS, των περιφερειακών και των GPIO pins που απαιτούνται από την εφαρμογή και, βέβαια, του ίδιου του BLE controller. Μετά το τέλος των αρχικοποιήσεων ο έλεγχος περνάει στο main loop του προγράμματος, όπου και παραμένει μέχρι να γίνει επανεκκίνηση του συστήματος. Η δομή του main loop περιγράφεται συνοπτικά ως εξής:

```
while (1)
{
    schedule();

    GLOBAL_INT_STOP(); // Disable interrupts

    // Get sleep mode
    sleep_mode = ble_sleep();

    // Enter sleep mode
    if (sleep_mode == mode_ext_sleep
        || sleep_mode == mode_deep_sleep)
    {
        set_radio_off();
        enable_cpu_deep_sleep();

        // Extended sleep mode
        if (sleep_mode == mode_ext_sleep)
        {
            set_pd_ext_sleep(); // Turn off Power Domains
                                // for extended sleep
        }
        // Deep sleep mode
        else
        {
            set_pd_deep_sleep(); // Turn off Power Domains
                                 // for deep sleep
        }

        WFI(); // SLEEP - Wait for interrupt

        disable_cpu_deep_sleep();
    }
    // No sleep
    else
    {
        WFI(); // Wait for interrupt
    }

    GLOBAL_INT_START(); // Enable interrupts
}
```

Το πρώτο πράγμα που γίνεται σε κάθε επανάληψη του main loop είναι η αποστολή των μηνυμάτων (schedule), τα οποία έχουν σταλεί από tasks και είναι pending στον kernel, με την κλήση των αντίστοιχων handlers στα tasks για τα οποία προορίζονται. Εδώ πρέπει να αναφέρουμε ότι, επειδή ο kernel, προκειμένου να παρέχει την υπηρεσία ανταλλαγής μηνυμάτων, χρησιμοποιεί πληροφορίες χρονισμού από το BLE baseband, αν το τελευταίο είναι κλειστό, τότε δεν μπορεί να εκτελεστεί η αποστολή μηνυμάτων. Επίσης, δεν επιτρέπεται ούτε να σταλούν νέα μηνύματα. Έτσι, έχουμε τον διαχωρισμό της λειτουργίας του συστήματος σε δύο φάσεις, τη σύγχρονη φάση, κατά την οποία το BLE baseband είναι ενεργοποιημένο και πραγματοποιείται η αποστολή και λήψη μηνυμάτων, και την ασύγχρονη φάση, κατά την οποία δεν επιτρέπεται ανταλλαγή μηνυμάτων.

Αυτό σημαίνει ότι, αν μια εφαρμογή βασίζεται σε ασύγχρονα γεγονότα, για παράδειγμα interrupts, δεν μπορεί να εκτελεί, στους αντίστοιχους handlers, διαδικασίες που απαιτούν συγχρονισμό, όπως η αποστολή μηνυμάτων. Αντίθετα, πρέπει να συγχρονίζει αυτά τα γεγονότα (π.χ. θέτοντας κάποιες σημαίες), ώστε οι απαιτούμενες λειτουργίες να εκτελούνται κατά τη σύγχρονη φάση. Για τον σκοπό αυτό, περιέχονται στο main loop μια σειρά από hooks, με τη βοήθεια των οποίων, μπορεί η εφαρμογή να συγχρονίσει τα διάφορα γεγονότα που συμβαίνουν. Ταυτόχρονα, τα hooks αυτά δίνουν τη δυνατότητα στην εφαρμογή να ελέγξει την είσοδο του συστήματος σε sleep mode. Μπορεί, δηλαδή, η εφαρμογή, με βάση την τρέχουσα κατάσταση στην οποία βρίσκεται (π.χ. έχει συμβεί κάποιο γεγονός που απαιτεί χειρισμό), να αποφασίσει ότι δεν επιθυμεί να περάσει σε sleep mode.

Μετά την αποστολή όλων των pending μηνυμάτων από τον kernel, το σύστημα ξεκινά τις διαδικασίες εισόδου σε sleep mode, ελέγχοντας πρώτα αν όντως μπορεί να περάσει σε αυτό. Για παράδειγμα, αν κάποιος timer αναμένεται να λήξει ή πρέπει να εκτελεστεί κάποια διαδικασία του BLE σε σύντομο χρονικό διάστημα, το σύστημα μπορεί να αποφασίσει ότι δεν έχει νόημα η ενεργοποίηση του sleep mode. Στη συνέχεια, αν η είσοδος σε sleep mode είναι εφικτή και επιτραπεί και από την εφαρμογή, το σύστημα αρχίζει να κλείνει τα power domains των διαφόρων στοιχείων, όπως BLE baseband, περιφερειακά, μνήμες, ανάλογα με το είδος του sleep mode (Extended ή Deep). Τελικά, τοποθετεί και τον επεξεργαστή σε κατάσταση deep sleep και σταματά περιμένοντας για κάποιο interrupt. Η ενεργοποίηση του συστήματος γίνεται από κάποιο από τα στοιχεία, τα οποία παραμένουν πάντα ανοιχτά και προορίζονται για αυτόν τον σκοπό. Η ενεργοποίηση μπορεί να γίνει είτε από κάποιο εξωτερικό γεγονός, είτε με τη λήξη ενός timer, που τίθεται από το σύστημα πριν την είσοδο σε sleep mode (π.χ. στο χρονικό διάστημα που μεσολαβεί μέχρι το επόμενο advertising ή connection event).

Τα hooks με τα οποία η εφαρμογή μπορεί να ελέγξει όλη την παραπάνω διαδικασία είναι τα ακόλουθα:

app_asynch_trm()	Καλείται κατά τη σύγχρονη φάση της λειτουργίας του συστήματος, αμέσως μετά τη schedule. Μπορεί να χρησιμοποιηθεί από την εφαρμογή για τον συγχρονισμό γεγονότων που ενεργοποιούνται ασύγχρονα. Σε αυτό το σημείο, η εφαρμογή μπορεί να στείλει νέα μηνύματα και να προκαλέσει νέα κλήση της schedule.
app_asynch_proc()	Καλείται στην αρχή του main loop και πριν την απενεργοποίηση των διακοπών. Το σύστημα μπορεί να βρίσκεται σε σύγχρονη ή ασύγχρονη φάση. Χρησιμεύει για την εκτέλεση διαδικασιών που προκύπτουν από interrupts. Τα interrupt handlers είναι καλό να διατηρούνται όσο το δυνατόν μικρότερα. Αν απαιτείται επιπλέον επεξεργασία, αυτή μπορεί να γίνει εδώ. Αν απαιτείται συγχρονισμός, η εφαρμογή μπορεί να ενεργοποιήσει το baseband, και να προκαλέσει νέα κλήση της schedule και της app_asynch_trm.
app_asynch_sleep_proc()	Καλείται μετά την app_asynch_proc και αφού έχουν απενεργοποιηθεί οι διακοπές. Είναι το τελευταίο σημείο στο οποίο η εφαρμογή μπορεί να ελέγξει για ασύγχρονα γεγονότα, πριν την εκκίνηση των διαδικασιών εισόδου σε sleep mode. Σε αυτό το σημείο η εφαρμογή μπορεί, βασισμένη στην τρέχουσα κατάστασή της, να ακυρώσει το sleep mode.
app_sleep_prepare_proc()	Καλείται μετά την απενεργοποίηση του BLE baseband, αλλά πριν την απενεργοποίηση των υπολοίπων power domains. Μπορεί να χρησιμοποιηθεί από την εφαρμογή για την ακύρωση του sleep mode.

app_sleep_entry_proc()	Καλείται ακριβώς πριν από την είσοδο σε sleep mode. Μπορεί να χρησιμοποιηθεί από την εφαρμογή για την εκτέλεση τυχόν διαδικασιών που απαιτούνται.
app_sleep_exit_proc()	Καλείται ακριβώς μετά από την έξοδο από το sleep mode. Μπορεί να χρησιμοποιηθεί από την εφαρμογή για την εκτέλεση τυχόν διαδικασιών που απαιτούνται.

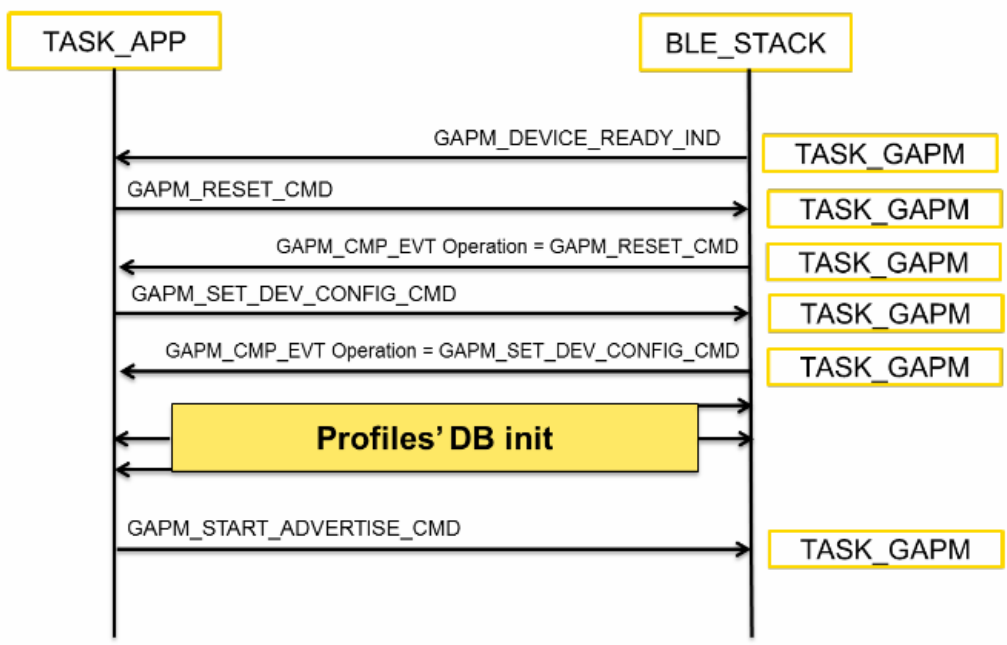
4.5.2 Επικοινωνία μεταξύ εφαρμογής και BLE stack

Τα διάφορα modules του firmware αρχικοποιούνται κατά την εκκίνηση του συστήματος. Καθένα από αυτά, δημιουργεί, όπως είδαμε παραπάνω, ένα ή περισσότερα tasks, τα οποία επικοινωνούν μεταξύ τους και με την εφαρμογή, ανταλλάσσοντας μηνύματα μέσω του kernel. Η εφαρμογή έχει το δικό της task, το οποίο και δημιουργεί πριν από την είσοδο στο main loop. Έτσι, κατά την κλήση της schedule, η εφαρμογή μπορεί να δεχτεί μηνύματα από τα modules του BLE stack.

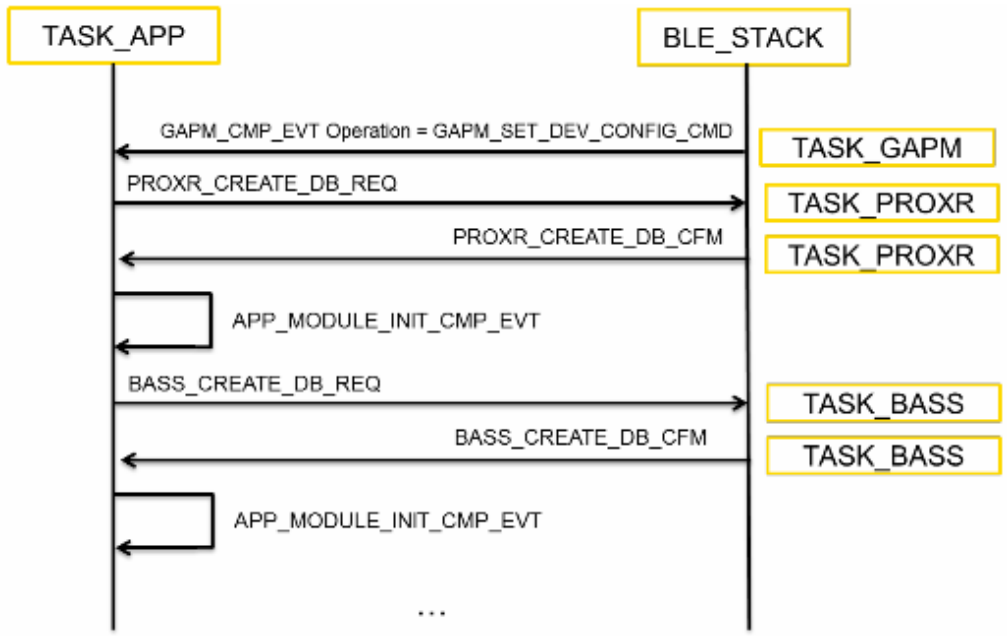
Το πρώτο από αυτά είναι το GAPM_DEVICE_READY_IND από τον GAP manager, το οποίο στέλνεται αμέσως μετά την εκκίνηση του συστήματος, όταν ολοκληρωθούν οι διαδικασίες αρχικοποίησης των κατώτερων επίπεδων. Η εφαρμογή απαντά, στέλνοντας το μήνυμα GAPM_RESET_CMD με παράμετρο GAPM_RESET, το οποίο και προκαλεί την επαναφορά όλων των υποσυστημάτων στις default επιλογές. Έτσι, εξασφαλίζεται ότι το σύστημα είναι σε μία πλήρως καθορισμένη και γνωστή κατάσταση, κατά την εκκίνηση της εφαρμογής. Μετά την ολοκλήρωση του RESET, ο GAP manager ενημερώνει την εφαρμογή με κατάλληλο μήνυμα. Στη συνέχεια, η εφαρμογή ρυθμίζει τις παραμέτρους του GAP στέλνοντας το μήνυμα GAPM_SET_DEV_CONFIG_CMD. Στα δεδομένα του μηνύματος περιέχονται ρυθμίσεις όπως ο GAP ρόλος της συσκευής, η τιμή του Appearance characteristic, το MTU, καθώς και οι επιθυμητές παράμετροι σύνδεσης στην περίπτωση που η συσκευή είναι peripheral. Μετά την ολοκλήρωση της ρύθμισης, η εφαρμογή αρχικοποιεί τη βάση δεδομένων του ATT, στέλνοντας κατάλληλα μηνύματα στα tasks των profile που είναι ενεργοποιημένα. Κάθε profile δημιουργεί τα δικά του services και ενημερώνει την εφαρμογή για την, επιτυχή ή μη, ολοκλήρωση της διαδικασίας (APP_MODULE_INIT_CMP_EVT).

Με το τέλος της αρχικοποίησης του επιπέδου ATT, η συσκευή, αν πρόκειται για peripheral, είναι πλέον έτοιμη να κάνει γνωστή την παρουσία της σε ενδιαφερόμενους clients. Έτσι, ξεκινά τη διαδικασία advertising στέλνοντας στον GAP manager το μήνυμα GAPM_START_ADVERTISE_CMD. Αν η συσκευή είναι τύπου central, τότε μπορεί να ξεκινήσει την αναζήτηση συσκευών με το μήνυμα GAPM_START_SCAN_CMD. Η λειτουργία του συστήματος συνεχίζεται με παρόμοιο τρόπο, δηλαδή με αποστολές μηνυμάτων, είτε από την πλευρά της εφαρμογής, προκειμένου να εκτελεστούν λειτουργίες του πρωτοκόλλου, είτε από την πλευρά του BLE stack, για την ενημέρωση της εφαρμογής για γεγονότα που έχουν συμβεί (π.χ. σύνδεση, αποσύνδεση).

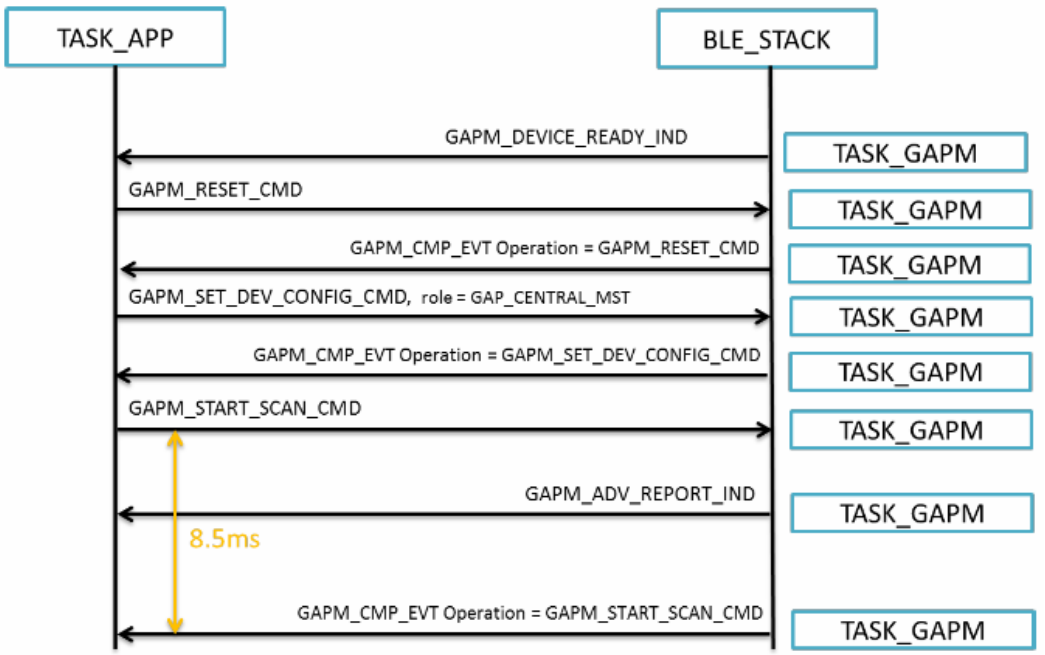
Οι παραπάνω ανταλλαγές μηνυμάτων, κατά την αρχικοποίηση της εφαρμογής, παρουσιάζονται σχηματικά στις ακόλουθες εικόνες:



Αρχικοποίηση ενός peripheral GATT server.



Αρχικοποίηση της βάσης δεδομένων του Attribute Protocol από τα Profiles.



Αρχικοποίηση ενός central GATT client.

4.5.3 Application Hooks

Οι διαδικασίες, που αναφέρονται παραπάνω, εκτελούνται από το application framework, που παρέχεται με το SDK του DA14580. Για τη σύνδεση του framework με την πραγματική εφαρμογή, που υλοποιεί τη συγκεκριμένη περίπτωση χρήσης για την οποία προορίζεται η συσκευή, ορίζεται ένα σύνολο από hook συναρτήσεις, οι οποίες καλούνται από το framework στα σημεία εκείνα, όπου απαιτείται από την εφαρμογή να θέσει τις επιθυμητές για αυτήν παραμέτρους και δεδομένα λειτουργίας του συστήματος. Ένας developer μπορεί, προσθέτοντας κώδικα στα κατάλληλα σημεία, να υλοποιήσει τη λειτουργικότητα της BLE εφαρμογής του. Οι σημαντικότερες hook συναρτήσεις που ορίζονται είναι οι ακόλουθες (η συμπεριφορά της εφαρμογής που περιγράφεται είναι αυτή ενός peripheral GATT server, όπως το δικό μας σύστημα):

periph_init	Αρχικοποίηση περιφερειακών κατά την εκκίνηση του συστήματος και μετά την έξοδο από sleep mode.
app_init_func	Αρχικοποίηση της εφαρμογής, πριν την είσοδο στο main loop. Η εφαρμογή αρχικοποιεί τα ενεργοποιημένα προφίλ.
app_configuration_func	Καλείται πριν την αποστολή του μηνύματος ρύθμισης του GAP manager. Η εφαρμογή θέτει τις απαιτούμενες ρυθμίσεις.
app_set_dev_config_complete_func	Καλείται με την ολοκλήρωση της ρύθμισης του GAP. Η εφαρμογή ξεκινά τη διαδικασία αρχικοποίησης της βάσης δεδομένων του ATT.
app_db_init_func	Αρχικοποίηση της βάσης δεδομένων του ATT. Η εφαρμογή στέλνει κατάλληλα μηνύματα στα ενεργοποιημένα προφίλ.

app_db_init_complete_func	Ολοκλήρωση της αρχικοποίησης της βάσης δεδομένων του ATT. Η εφαρμογή μπορεί να ξεκινήσει το advertising.
app_adv_func	Καλείται πριν ξεκινήσει το advertising. Η εφαρμογή θέτει τις παραμέτρους του advertising, καθώς και τα advertising και scan response data.
app_connection_func	Καλείται κατά τη σύνδεση με μια απομακρυσμένη συσκευή. Η εφαρμογή εκτελεί τις απαραίτητες λειτουργίες, ενημερώνει τα ενεργοποιημένα προφίλ και περνάει σε κατάσταση σύνδεσης.
app_disconnect_func	Καλείται κατά την αποσύνδεση από την απομακρυσμένη συσκευή. Η εφαρμογή εκτελεί τις απαραίτητες λειτουργίες και μπορεί να ενημερώσει τα ενεργοποιημένα προφίλ (κάθε προφίλ λαμβάνει επίσης μήνυμα αποσύνδεσης). Στη συνέχεια ξεκινά το advertising.
app_sec_init_func	Αρχικοποίηση του επίπεδου ασφάλειας που απαιτείται από την εφαρμογή.

Στην περίπτωση που η εφαρμογή ενεργοποιήσει κάποιο επίπεδο ασφαλείας, πρέπει να υλοποιήσει μια σειρά από επιπλέον hook συναρτήσεις. Αυτές χρησιμοποιούνται για λειτουργίες, όπως pairing, bonding, δημιουργία και ανταλλαγή των απαραίτητων κλειδιών, εκκίνηση κρυπτογράφησης.

4.6 Profiles

Τα προφίλ είναι από τα σημαντικότερα στοιχεία μιας εφαρμογής Bluetooth Low Energy. Ορίζουν τους ρόλους της κάθε περίπτωσης χρήσης, συγκεκριμένες συμπεριφορές που αναμένονται από τις συμβατές συσκευές, τις υπηρεσίες που πρέπει να υποστηρίζονται και τα χαρακτηριστικά που αυτές περιέχουν. Είναι ουσιαστικά το πρότυπο, που επιτρέπει στις συσκευές που το υποστηρίζουν, να ανταλλάσσουν δεδομένα χρησιμοποιώντας το BLE, προκειμένου να υλοποιήσουν τη λειτουργικότητα που προβλέπεται από το προφίλ. Μια εφαρμογή BLE υλοποιεί ένα ή περισσότερα προφίλ, με σκοπό να προσφέρει αυτή τη λειτουργικότητα, σε συνδυασμό με άλλες, συμβατές με το προφίλ, συσκευές. Τα προφίλ αυτά μπορεί να είναι ορισμένα από το Bluetooth SIG (adopted profiles) ή custom profiles, ορισμένα από τον developer της εφαρμογής.

Εξαιτίας της σπουδαιότητας αυτής των προφίλ στη δημιουργία BLE εφαρμογών, το SDK του DA14580, παρέχει μια σειρά από ευκολίες για τον ορισμό προφίλ. Επίσης, περιέχει πολλά από τα SIG adopted profiles με τη μορφή κώδικα, τα οποία μπορούν να ενεργοποιηθούν με χρήση κατάλληλων preprocessor macros και προσθήκη του απαραίτητου κώδικα στα application hooks.

Το SDK χωρίζει κάθε προφίλ σε δύο τμήματα. Το πρώτο είναι το ανεξάρτητο της εφαρμογής τμήμα, το οποίο υλοποιεί λειτουργίες όπως η δημιουργία των υπηρεσιών στη βάση δεδομένων του ATT, η ενεργοποίηση ή απενεργοποίησή τους, αντίστοιχα, κατά τη σύνδεση ή την αποσύνδεση, και η υποστήριξη γενικών λειτουργιών του προφίλ. Μια από τις σημαντικότερες λειτουργίες του προφίλ, που υλοποιούνται από το συγκεκριμένο τμήμα, είναι οι διαδικασίες που συμβαίνουν, όταν ένας client γράφει την τιμή κάποιου χαρακτηριστικού στις υπηρεσίες που παρέχονται από το συγκεκριμένο προφίλ. Ο κώδικας υλοποίησης του προφίλ ελέγχει το χαρακτηριστικό, του οποίου την τιμή θέλει να μεταβάλει ο client, και

εκτελεί τις διαδικασίες που προβλέπει το προφίλ σε αυτή την περίπτωση. Για παράδειγμα, η εγγραφή αυτή μπορεί να αλλάζει παραμέτρους λειτουργίας του συστήματος. Επίσης, αν πρόκειται για control-point χαρακτηριστικό, η εγγραφή συνήθως εκλαμβάνεται ως εντολή, που εκκινεί μια συγκεκριμένη λειτουργία στη συσκευή, η οποία λειτουργία προβλέπεται και πάλι από το προφίλ.

Το δεύτερο τμήμα είναι το εξαρτώμενο από την εφαρμογή, το οποίο υλοποιεί λειτουργίες και συμπεριφορές που περιγράφονται μεν στο προφίλ, αλλά με γενικό και αφαιρετικό τρόπο. Ο τρόπος υλοποίησης των λειτουργιών αυτών εξαρτάται από το είδος της συσκευής και τη συγκεκριμένη περίπτωση χρήσης. Ο κώδικας υλοποίησης του προφίλ ορίζει τον τρόπο που επιτυγχάνεται η επικοινωνία των δύο τμημάτων, ώστε ο συνδυασμός τους να υλοποιεί την αναμενόμενη συμπεριφορά.

Για να γίνει περισσότερο κατανοητό το παραπάνω, χρησιμοποιούμε το παράδειγμα του Proximity Profile Reporter role, το οποίο παρέχεται από το SDK. Το ανεξάρτητο της εφαρμογής τμήμα αρχικοποιεί, στο ATT database, τα τρία services που προβλέπονται στο προφίλ: Link Loss Service (LLS), Immediate Alert Service (IAS) και Tx Power Service (TPS). Σύμφωνα με το προφίλ, αν διακοπεί η σύνδεση μεταξύ των δύο συσκευών, ή αν το Monitor γράψει κατάλληλη τιμή στο Alert Level του IAS (ως αποτέλεσμα της αύξησης της απόστασης των συσκευών πέρα από κάποιο όριο), τότε η συσκευή Reporter, πρέπει να εκκινήσει τη λειτουργία ειδοποίησης (alert). Το πώς υλοποιείται αυτή η ειδοποίηση δεν ορίζεται στο προφίλ, και αποτελεί, στην προκειμένη περίπτωση, μέρος του εξαρτώμενου από την εφαρμογή τμήματος. Έτσι, αφού το ανεξάρτητο της εφαρμογής τμήμα ειδοποιηθεί από το BLE stack για την αποσύνδεση ή την εγγραφή στο Alert Level του IAS, ειδοποιεί με τη σειρά του το άλλο τμήμα του κώδικα, το οποίο και ξεκινά το alert, με τον τρόπο που προβλέπεται για τη συγκεκριμένη συσκευή (π.χ. παραγωγή ήχου, άναμμα/σβήσιμο LED).

Το ανεξάρτητο της εφαρμογής τμήμα ορίζει ένα task με ένα instance ανά σύνδεση. Μέσω του task αυτού και των message handlers που ορίζει, μπορεί το τμήμα αυτό να φέρει εις πέρας τις λειτουργίες που αναφέραμε παραπάνω, ανταλλάσσοντας μηνύματα με το δεύτερο τμήμα ή το BLE stack. Το δεύτερο τμήμα δεν έχει δικό του task, αλλά χρησιμοποιεί εκείνο της εφαρμογής. Αποτελεί το interface μεταξύ της εφαρμογής και του πρώτου τμήματος. Η επικοινωνία μεταξύ των δύο τμημάτων επιτυγχάνεται με ανταλλαγή custom μηνυμάτων, και των αντίστοιχων δεδομένων, που ορίζονται από τον κώδικα υλοποίησης. Πρέπει να αναφέρουμε επίσης, ότι, κατά την αρχικοποίηση του συστήματος, το firmware του DA14580 καλεί μία συνάρτηση μέσω του jump table, η οποία, με τη σειρά της, καλεί τις συναρτήσεις αρχικοποίησης των ενεργοποιημένων προφίλ, για το ανεξάρτητο της εφαρμογής τμήμα. Οι τελευταίες έχουν ως βασική λειτουργία τη δημιουργία των αντίστοιχων tasks.

Για το παράδειγμα του Proximity Profile Reporter role, τα βασικότερα μηνύματα είναι τα εξής (όπου ως πρώτο τμήμα θεωρείται το ανεξάρτητο της εφαρμογής και ως δεύτερο το εξαρτώμενο από την εφαρμογή):

- **PROXR_CREATE_DB_REQ**: Στέλνεται από το δεύτερο τμήμα στο πρώτο, κατά τη διαδικασία της αρχικοποίησης του ATT database (app_db_init_func). Το πρώτο τμήμα δημιουργεί τα services και απαντά με το PROXR_CREATE_DB_CFM.

- **PROXR_CREATE_DB_CFM**: Στέλνεται από το πρώτο στο δεύτερο τμήμα, όταν ολοκληρωθεί η δημιουργία των services στο ATT database. Το δεύτερο τμήμα ειδοποιεί την εφαρμογή για την ολοκλήρωση, στέλνοντάς της ένα μήνυμα APP_MODULE_INIT_CMP_EVT.

- **PROXR_ENABLE_REQ**: Στέλνεται από το δεύτερο τμήμα στο πρώτο, κατά τη σύνδεση με μια απομακρυσμένη συσκευή (app_connection_func). Το πρώτο τμήμα ενεργοποιεί τα services.

- **GAPC_DISCONNECT_IND**: Στέλνεται από τον GAP controller της σύνδεσης στο πρώτο τμήμα, όταν συμβεί αποσύνδεση. Το πρώτο τμήμα ελέγχει τον λόγο της αποσύνδεσης, και, αν είναι απαραίτητο, ειδοποιεί το δεύτερο τμήμα μέσω του μηνύματος PROXR_ALERT_IND. Επίσης, απενεργοποιεί τα services.

- **GATTC_WRITE_CMD_IND**: Στέλνεται από τον GATT controller της σύνδεσης στο πρώτο τμήμα, όταν το Monitor γράφει στην τιμή κάποιου από τα χαρακτηριστικά που περιέχονται στο ATT database. Αν πρόκειται για το Alert Level του IAS, τότε το πρώτο τμήμα ειδοποιεί το δεύτερο μέσω του μηνύματος PROXR_ALERT_IND.

- **PROXR_ALERT_IND**: Στέλνεται από το πρώτο στο δεύτερο τμήμα, προκειμένου αυτό να εκκινήσει τη διαδικασία του alert, όπως ορίζει το προφίλ και με τον τρόπο που υποστηρίζεται από τη συγκεκριμένη συσκευή.

4.7 Εργαλεία

Για τη συγγραφή του κώδικα στο αναπτυξιακό DA14580, η οποία γίνεται στη γλώσσα προγραμματισμού C, χρησιμοποιήθηκε το περιβάλλον προγραμματισμού MDK-ARM (Microcontroller Development Kit), το οποίο αναπτύσσεται από την εταιρία Keil, που ανήκει στην ARM Holdings. Το MDK-ARM είναι ένα πλήρες περιβάλλον προγραμματισμού για ενσωματωμένα συστήματα, όπως το DA14580, τα οποία βασίζονται σε επεξεργαστές της ARM. Παρέχει ένα IDE, το μVision, το C/C++ Compilation Toolchain της ARM, καθώς και πλήθος εργαλείων που είναι απαραίτητα για τον προγραμματισμό, τη φόρτωση του κώδικα, τον έλεγχο και το debugging ενσωματωμένων εφαρμογών. Το μVision, εκτός από τη συγγραφή του κώδικα, λειτουργεί ως user interface και για τα υπόλοιπα εργαλεία, παρέχοντας δυνατότητες visual debugging και simulation environment.

Στο SDK του DA14580 περιέχονται εργαλεία για τον έλεγχο των εφαρμογών και των πρωτοτύπων, που υλοποιούνται με τη βοήθεια της αναπτυξιακής πλακέτας. Ένα από αυτά είναι ο ConnectionManager. Το πρόγραμμα αυτό, χρησιμοποιώντας το proprietary HCI του DA14580, μπορεί, αφού φορτώσει σε αυτό ένα firmware, να λειτουργήσει ως peripheral ή central και ως GATT server ή client. Δίνει δυνατότητες ορισμού πολλών παραμέτρων του BLE, ανάγνωσης και εγγραφής τιμών χαρακτηριστικών στην απομακρυσμένη συσκευή, ενώ, παράλληλα, παρέχει μια επισκόπηση των μηνυμάτων που ανταλλάσσονται κατά τη λειτουργία του συστήματος. Μπορεί να χρησιμοποιηθεί για τον έλεγχο του ATT database και των λειτουργιών του προφίλ μιας συσκευής, χωρίς να χρειάζεται να δημιουργηθεί συμβατός client για αυτήν. Ορίζει, επίσης, ένα Test Mode, με τη βοήθεια του οποίου μπορεί να εκτελεστεί μία σειρά από ελέγχους στο φυσικό επίπεδο της BLE συσκευής.

Ένα δεύτερο εργαλείο είναι το SmartSnippets, το οποίο παρέχει λειτουργίες για το χειρισμό της αναπτυξιακής πλακέτας και δυνατότητες επιπλέον ελέγχων στις ενσωματωμένες εφαρμογές που εκτελούνται σε αυτή. Βασικότερα χαρακτηριστικά του είναι η δυνατότητα εγγραφής στην OTP μνήμη του DA14580 και η δυνατότητα μέτρησης της κατανάλωσης ρεύματος του συστήματος κατά τη λειτουργία της εφαρμογής, ώστε η εφαρμογή να μπορεί να ελεγχθεί και ως προς αυτή την παράμετρο.

Κεφάλαιο 5

Προγραμματισμός σε Android

Android Logo

Για τη δημιουργία του τμήματος του συστήματος καταγραφής καιρού, το οποίο εκτελείται σε smartphone ή tablet, χρησιμοποιήσαμε το λειτουργικό σύστημα Android. Για τον λόγο αυτό, αναφέρουμε σε αυτό το κεφάλαιο κάποια βασικά στοιχεία για τη συγκεκριμένη πλατφόρμα και τον προγραμματισμό σε αυτή. Ο προγραμματισμός σε Android είναι, βέβαια, ένα πολύ ευρύ πεδίο, το οποίο είναι αδύνατο να καλυφθεί μέσα σε ένα κεφάλαιο. Αυτό, που θα προσπαθήσουμε να κάνουμε, είναι να παρουσιάσουμε κάποιες βασικές έννοιες και συστατικά του λειτουργικού συστήματος Android και των εφαρμογών που τρέχουν σε αυτό, ώστε να γίνει ευκολότερα κατανοητή η παρουσίαση της εφαρμογής σε επόμενο κεφάλαιο. Δε θα κάνουμε δηλαδή λεπτομερειακή περιγραφή της πλατφόρμας, ούτε θα αναφέρουμε στοιχεία που δε χρησιμοποιήσαμε. Τέλος, θα κάνουμε μια παρουσίαση του Bluetooth Low Energy API, που παρέχεται από το Android στους developers BLE εφαρμογών, το οποίο και αποτελεί το βασικό στοιχείο, πάνω στο οποίο χτίσαμε την εφαρμογή.

5.1 Εισαγωγή – Ιστορικά στοιχεία

Το Android είναι ένα λειτουργικό σύστημα, βασισμένο στον πυρήνα του Linux, που απευθυνόταν αρχικά σε κινητές συσκευές. Αναπτύσσεται από την εταιρία Google και είναι σήμερα το δημοφιλέστερο λειτουργικό σύστημα για κινητές συσκευές, με το μεγαλύτερο install base, ανάμεσα στις ανταγωνιστικές πλατφόρμες. Εκτός από τις κινητές συσκευές, το Android χρησιμοποιείται ως λειτουργικό σύστημα σε μια ευρεία γκάμα προϊόντων, όπως τηλεοράσεις (smart-TV), media players, παιχνιδομηχανές, ακόμα και σε αυτοκίνητα και ρολόγια (smart-watch).

Η εταιρία Android, Inc. δημιουργήθηκε τον Οκτώβριο του 2003 στο Palo Alto της California. Αρχικός σκοπός της ήταν η δημιουργία ενός εξελιγμένου λειτουργικού συστήματος για ψηφιακές φωτογραφικές μηχανές. Διαπιστώνοντας, όμως, ότι η συγκεκριμένη αγορά δεν ήταν αρκετά μεγάλη, οι ιδρυτές της εταιρίας αποφάσισαν να προχωρήσουν στη δημιουργία ενός λειτουργικού συστήματος για smartphones. Εκείνη την εποχή, τα smartphones έτρεχαν λειτουργικά συστήματα όπως το Symbian της Nokia και τα

Windows Mobile της Microsoft. Η εταιρία, παρότι προχωρούσε στην ανάπτυξη του συστήματος, δεν έβγαλε τελικά κάποιο προϊόν στην αγορά. Τον Αύγουστο του 2005, η εταιρία αγοράστηκε από την Google, μια κίνηση που θεωρήθηκε ως προσπάθεια της Google να εισέλθει στην αγορά των κινητών τηλεφώνων. Η ανάπτυξη του συστήματος συνεχίστηκε και τα επόμενα χρόνια, αλλά και πάλι, εκτός από κάποια prototypes, δεν εμφανίστηκε κάποιο προϊόν στην αγορά.

Τελικά, το Νοέμβριο του 2007, δημιουργήθηκε το Open Handset Alliance, μια ένωση από εταιρίες όπως η Google, κατασκευαστές κινητών τηλεφώνων όπως οι HTC, Samsung και Sony, πάροχοι κινητής τηλεφωνίας, καθώς και κατασκευαστές hardware όπως η Qualcomm και η Texas Instruments. Το ίδιο έτος, λίγο νωρίτερα, είχε κάνει την εμφάνισή του το iPhone της Apple και η αγορά των smartphones, που αφορούσε παλαιότερα μόνο ειδικές περιπτώσεις προχωρημένων χρηστών, είχε ήδη αρχίσει να διευρύνεται. Στόχος του Open Handset Alliance, ήταν η δημιουργία και προώθηση ανοιχτών προτύπων για κινητές συσκευές. Ως πρώτο προϊόν του, παρουσιάστηκε το λειτουργικό σύστημα Android, μία πλατφόρμα ανοιχτού κώδικα για κινητές συσκευές, βασισμένη πάνω στον πυρήνα 2.6.25 του Linux. Περίπου ένα χρόνο αργότερα, εμφανίστηκε στην αγορά η πρώτη συσκευή smartphone με το λειτουργικό σύστημα Android, το HTC Dream.

Η ανοιχτή φύση της πλατφόρμας την έκανε αρκετά δημοφιλή στους κατασκευαστές κινητών τηλεφώνων και, έτσι, όλο και περισσότερες συσκευές άρχισαν να βγαίνουν στην αγορά, ενώ η δημοτικότητα του Android στους τελικούς χρήστες αυξανόταν με γρήγορους ρυθμούς. Το 2010, η Google ανακοίνωσε τη δημιουργία του πρώτου «δικού» της τηλεφώνου, με τη σειρά Nexus. Η πρώτη συσκευή της σειράς ήταν το Nexus One της HTC. Η σειρά Nexus ανανεώνεται τακτικά με νέες συσκευές smartphone και tablet, οι οποίες κατασκευάζονται από την Google σε συνεργασία με κάποια κατασκευάστρια εταιρία. Τα τελευταία μοντέλα της σειράς είναι το smartphone Nexus 5 της LG και τα tablet Nexus 7 της Asus και Nexus 10 της Samsung. Τα Nexus smartphones και tablets θεωρούνται reference designs για το Android, αφού κάθε νέο προϊόν της σειράς κυκλοφορεί, συνήθως, με μια καινούρια έκδοση του λειτουργικού, και χρησιμοποιείται για την επίδειξη των νέων δυνατοτήτων της πλατφόρμας.

Παράλληλα, η Google δημιούργησε το Android Market, το οποίο αργότερα μετονομάστηκε σε Play Store, μέσα από το οποίο οι developers έχουν τη δυνατότητα να διανέμουν τις εφαρμογές τους για την πλατφόρμα του Android, και οι τελικοί χρήστες να βρίσκουν και να κατεβάζουν νέες εφαρμογές, που καλύπτουν ένα ευρύ φάσμα περιπτώσεων χρήσης. Μέχρι σήμερα, πολλά δισεκατομμύρια εφαρμογές έχουν αγοραστεί ή κατεβαστεί από αυτό.

5.2 Εκδόσεις του Android

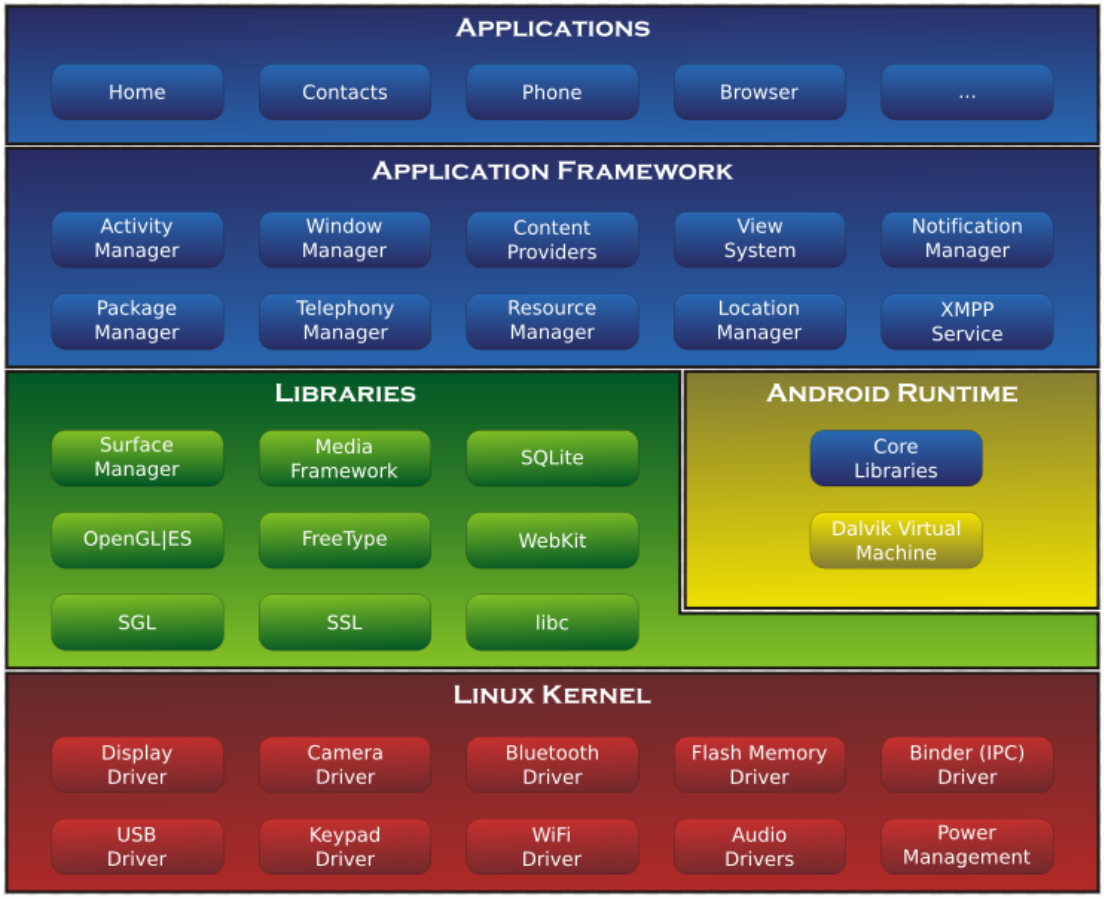
Το Android αναπτύσσεται ιδιωτικά από την Google μέχρι οι τελευταίες αλλαγές, ενημερώσεις και νέες δυνατότητες να είναι έτοιμες προς κυκλοφορία. Σε αυτό το σημείο, με την κυκλοφορία της νέας έκδοσης, ο κώδικας του λειτουργικού συστήματος γίνεται διαθέσιμος υπό της άδεια Apache License 2.0. Αυτό δεν ισχύει για τον πυρήνα του Linux, που έχει τη δική του άδεια χρήσης GNU GPL v2, και οι αλλαγές είναι άμεσα διαθέσιμες. Η πλατφόρμα του Android, παρότι είναι ανοιχτή, περιέχει και μια σειρά από proprietary οδηγούς και εφαρμογές. Αυτό ισχύει ακόμα και για κάποιες από τις βασικότερες εφαρμογές του συστήματος, που αναπτύσσονται από την ίδια την Google.

Το Android βρίσκεται σήμερα στην έκδοση 4.4.4, το τελευταίο update της έκδοσης 4.4 με την κωδική ονομασία KitKat. Ακολουθεί ένας πίνακας με τις κυριότερες εκδόσεις που έχουν κυκλοφορήσει μέχρι σήμερα, και τα αντίστοιχα API level, δηλαδή τις δυνατότητες που παρέχονται στους developers, που υποστηρίζονται από αυτές.

Έκδοση	**Κωδική ονομασία**	**Ημερομηνία κυκλοφορίας**	**API Level**
1.0	n/a	23-9-2008	1
1.1	n/a	27-2-2009	2
1.5	n/a	27-4-2009	3
1.6	Donut	15-9-2009	4
2.0	Eclair	26-9-2009	5
2.1	Eclair	12-1-2010	7
2.2	Froyo	20-5-2010	8
2.3	Gingerbread	6-12-2010	9
3.0	Honeycomb	22-2-2011	11
3.1	Honeycomb	10-5-2011	12
3.2	Honeycomb	15-7-2011	13
4.0	Ice Cream Sandwich	18-10-2011	14
4.1	Jelly Bean	9-7-2012	16
4.2	Jelly Bean	13-11-2012	17
4.3	Jelly Bean	24-7-2013	18
4.4	KitKat	31-10-2013	19

Η υποστήριξη του Bluetooth Low Energy ξεκίνησε με την έκδοση 4.3 και το API Level 18.

5.3 Αρχιτεκτονική και προγραμματισμός του συστήματος



Στην παραπάνω εικόνα, βλέπουμε συνοπτικά τα κυριότερα στοιχεία του λειτουργικού συστήματος Android. Στο κατώτερο επίπεδο έχουμε τον πυρήνα του Linux και τους οδηγούς των συσκευών. Ο πυρήνας ελέγχει την πρόσβαση στο υλικό και παρέχει στα ανώτερα επίπεδα τις απαραίτητες για τη λειτουργία τους υπηρεσίες (processes, threads, memory management, IO, timers, locks κλπ). Πάνω από αυτό βρίσκονται οι βασικές βιβλιοθήκες του συστήματος, όπως η runtime βιβλιοθήκη της C (libc), η βιβλιοθήκη του OpenGL για επιτάχυνση γραφικών και η βιβλιοθήκη της SQLite για πρόσβαση σε βάσεις δεδομένων αυτού του τύπου. Παράλληλα έχουμε το Android Runtime, το οποίο αποτελείται από την εικονική μηχανή Dalvik για Java και τα Core Libraries. Η Dalvik είναι μία εικονική μηχανή για Java bytecode, η οποία είναι βελτιστοποιημένη για χρήση σε κινητές και ενσωματωμένες συσκευές. Χρησιμοποιεί just-in-time (JIT) compilation και ένα δικό της format για τα εκτελέσιμα αρχεία, το λεγόμενο dex (Dalvik Executable). Με τις τελευταίες εκδόσεις του Android, η Google κινείται προς την αντικατάσταση της Dalvik με το Android Runtime (ART), το οποίο θα τρέχει επίσης Java bytecode, χρησιμοποιώντας όμως τεχνικές ahead-of-time (AOT) compilation.

Η γλώσσα Java αναπτύχθηκε, αρχικά, από την εταιρία Sun, ενώ σήμερα, μετά την εξαγορά της τελευταίας, ανήκει και αναπτύσσεται από την Oracle. Η Java απέκτησε αρκετή δημοφιλία στα πρώτα χρόνια του World Wide Web, με τη δυνατότητα δημιουργίας μικρο-εφαρμογών (τα λεγόμενα applets), που έτρεχαν στον browser του χρήστη. Αν και για αυτή τη χρήση έχει, πλέον, αντικατασταθεί από άλλες τεχνολογίες, η Java συνεχίζει να χρησιμοποιείται σε πλήθος εφαρμογών για απλούς χρήστες και σε εταιρικά περιβάλλοντα, με

πολλές τεχνολογίες να βασίζονται σε αυτή. Είναι, επίσης και η βασική γλώσσα προγραμματισμού για το Android. Αν και είναι δυνατό να γραφτούν native εφαρμογές για το Android, με τη βοήθεια του Native Development Kit (NDK), ο προτιμώμενος και πιο εύκολος τρόπος δημιουργίας Android apps είναι σε Java. Η Google παρέχει τα απαραίτητα εργαλεία για αυτόν τον σκοπό, με τα Android SDK και Android Development Tools (ADT), με IDE το Eclipse, ενώ, τελευταία, προωθεί τη χρήση του Android Studio.

Οι συνηθισμένες εφαρμογές Android είναι, λοιπόν, εφαρμογές Java, που τρέχουν πάνω στην Dalvik Virtual Machine, χρησιμοποιώντας τις υπηρεσίες του Application Framework. Το τελευταίο παρέχει μια σειρά από managers, που ελέγχουν διάφορα στοιχεία του συστήματος. Για παράδειγμα έχουμε Location Manager για το GPS, Notification Manager για το σύστημα ειδοποιήσεων, Telephony Manager για την τηλεφωνία, Bluetooth Manager για το Bluetooth, Activity και Window Manager για το user interface.

Κάθε εφαρμογή τρέχει σε δική της διεργασία και εικονική μηχανή. Επίσης, το σύστημα δίνει σε κάθε εφαρμογή ένα τοπικό σύστημα αρχείων και ένα ξεχωριστό Linux user ID. Αυτό εξασφαλίζει ότι μία εφαρμογή δεν μπορεί, κατά λάθος ή επίτηδες, να επηρεάσει τη λειτουργία άλλων εφαρμογών. Έχουμε δηλαδή το απαραίτητο isolation μεταξύ των διαφόρων εφαρμογών. Ανάλογα με τα συστατικά της, τα οποία είναι ενεργά, και το αν αυτή είναι ορατή στον χρήστη, η διεργασία μιας εφαρμογής μπορεί να βρίσκεται σε μία από πέντε καταστάσεις, οι οποίες, με φθίνουσα σημασία, είναι οι: Foreground, Visible, Service, Background και Empty. Το σύστημα μπορεί να αποφασίσει να τερματίσει μια διεργασία, αν οι ελεύθεροι πόροι φτάσουν σε κρίσιμο σημείο. Η διεργασία και η εφαρμογή θα ξεκινήσουν ξανά όταν ο χρήστης επιστρέψει σε αυτές. Οι εφαρμογές πρέπει να είναι γραμμένες με τέτοιο τρόπο, ώστε αυτή η διαδικασία να μην επηρεάζει την εμπειρία του χρήστη.

5.4 Συστατικά Εφαρμογών Android

Οι εφαρμογές Android αποτελούνται από τέσσερις τύπους συστατικών (components):

- Activities
- Services
- Content Providers
- Broadcast Receivers

5.4.1 Activity

Ένα activity αντιπροσωπεύει μία οθόνη της εφαρμογής, η οποία παρουσιάζει στον χρήστη ένα user interface (UI). Είναι το κομμάτι της εφαρμογής, με το οποίο έρχεται σε άμεση επαφή ο χρήστης, πάνω στο οποίο μπορεί να εκτελεί ενέργειες και να βλέπει τα αποτελέσματά τους. Το UI του activity ορίζεται, συνήθως, με τη βοήθεια ενός αρχείου XML. Μπορεί να οριστεί και προγραμματιστικά, αλλά ο πρώτος τρόπος είναι προτιμότερος, αφού αποσυνδέει τον κώδικα της εφαρμογής από τη μορφή του UI και έχει και άλλα πλεονεκτήματα, όπως θα δούμε παρακάτω.

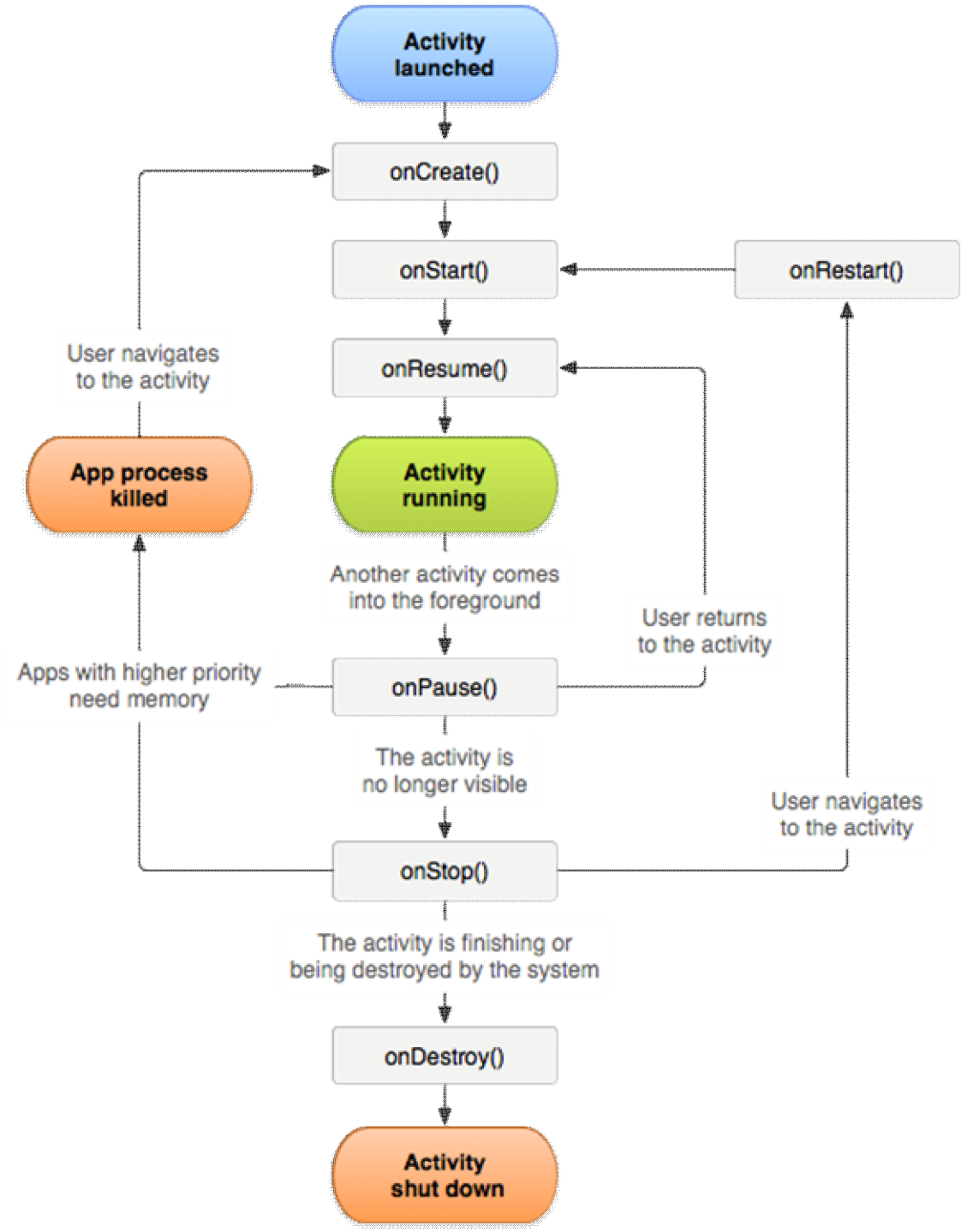
Ένα activity μπορεί να βρίσκεται σε μία από τρεις καταστάσεις:

- **Resumed (Running)**: Το activity βρίσκεται στο προσκήνιο και έχει το user focus.
- **Paused**: Ένα άλλο activity βρίσκεται στο προσκήνιο και έχει το user focus, αλλά αυτό το activity είναι ακόμα ορατό.
- **Stopped**: Το activity βρίσκεται στο παρασκήνιο και δεν είναι ορατό στον χρήστη.

Κατά τη διάρκεια του κύκλου ζωής του activity, το σύστημα καλεί μια σειρά από callback μεθόδους πάνω σε αυτό. Οι κυριότερες είναι οι εξής:

onCreate()	Καλείται κατά τη δημιουργία του activity. Ακολουθείται πάντα από την onStart.
onRestart()	Καλείται πριν την onStart, όταν το activity έχει γίνει Stopped.
onStart()	Καλείται αμέσως πριν το activity γίνει ορατό στον χρήστη.
onResume()	Καλείται όταν το activity είναι πλέον ορατό, προτού ο χρήστης αρχίσει να αλληλεπιδρά με αυτό. Ακολουθείται πάντα από την onPause.
onPause()	Καλείται όταν κάποιο άλλο activity πρόκειται να περάσει στο προσκήνιο.
onStop()	Καλείται όταν το activity δεν είναι πλέον ορατό στον χρήστη.
onDestroy()	Καλείται πριν την καταστροφή του activity.

Μετά την κλήση των onPause, onStop ή onDestroy, το activity θεωρείται killable και το σύστημα μπορεί να αποφασίσει να τερματίσει τη διεργασία του. Ο κύκλος ζωής ενός activity και οι αντίστοιχες μέθοδοι παρουσιάζονται στην εικόνα της διπλανής σελίδας.



Activity Lifecycle

5.4.2 Service

Ένα service είναι ένα στοιχείο μιας εφαρμογής, που χρησιμοποιείται, κυρίως, για την εκτέλεση λειτουργιών, που απαιτούν μεγάλο χρονικό διάστημα, στο παρασκήνιο. Σε αντίθεση με το activity, ένα service δεν παρέχει UI. Για την εκτέλεση χρονοβόρων ενεργειών, μπορούν επίσης να χρησιμοποιηθούν και Java threads. Όμως, σε αντίθεση με το service, ένα thread είναι συνδεδεμένο με το αντικείμενο που το δημιούργησε (π.χ. ένα activity), οπότε ο κύκλος ζωής του δεν μπορεί να ξεπεράσει αυτόν του πατρικού αντικειμένου.

Ένα service εμφανίζεται με δύο μορφές, ενώ μπορεί να βρίσκεται ταυτόχρονα και στις δύο:

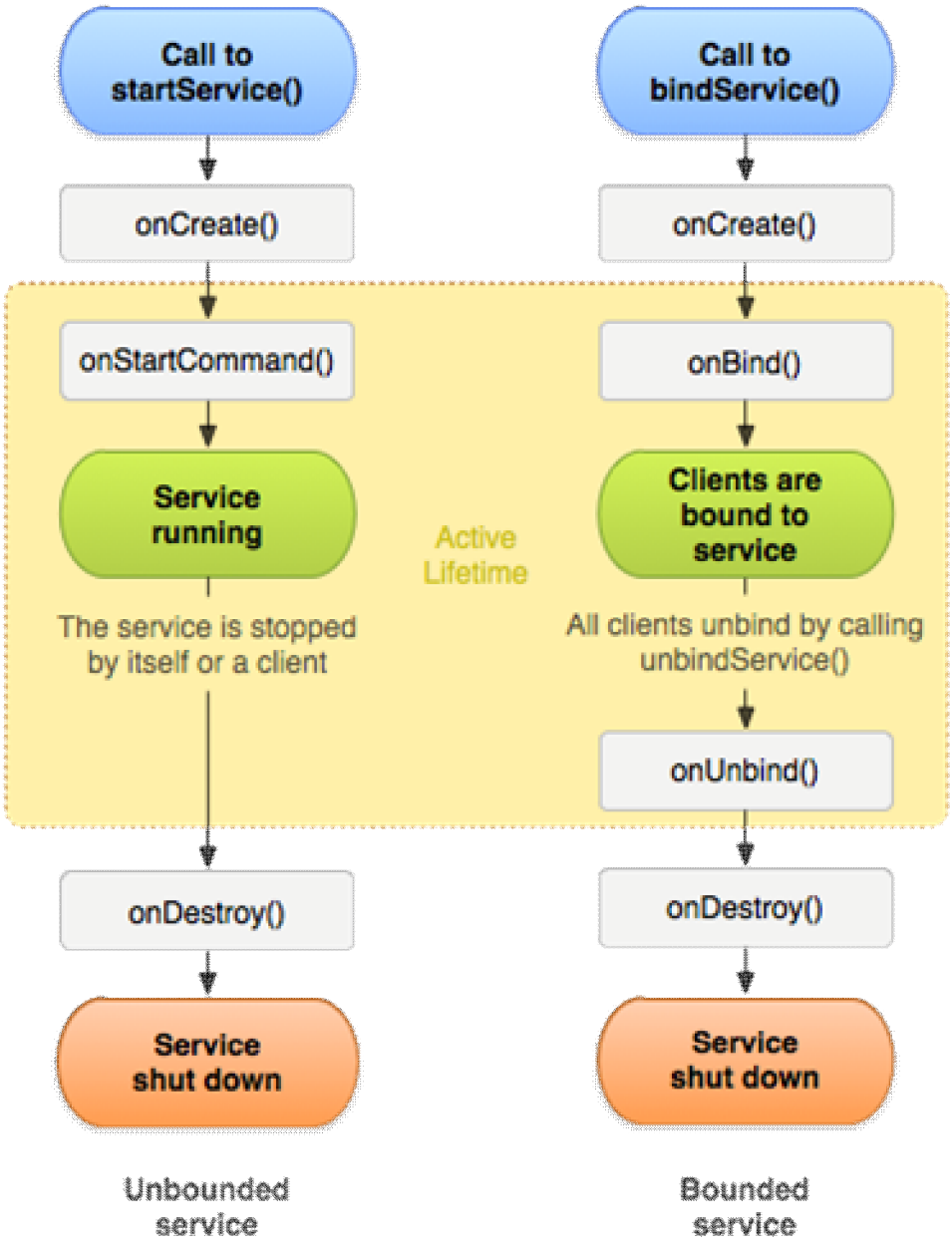
- **Started**: Ένα component μιας εφαρμογής μπορεί να ξεκινήσει το service, καλώντας τη μέθοδο startService(). Από εκεί και πέρα, το service μπορεί να τρέχει για απεριόριστο χρονικό διάστημα, ακόμα και αν το αρχικό component καταστραφεί. Το service συνεχίζει να εκτελείται, μέχρι κάποιο άλλο component να το σταματήσει, είτε να σταματήσει το ίδιο τον εαυτό του, αφού ολοκληρώσει την εργασία για την οποία ξεκίνησε.

- **Bound**: Ένα component μπορεί να συνδεθεί (bind) με ένα service, καλώντας τη μέθοδο bindService(). Ένα bound service παρέχει ένα interface, μέσω του οποίου, τα components που συνδέονται με αυτό, μπορούν να αλληλεπιδράσουν μαζί του. Τα components μπορεί να ανήκουν στην ίδια ή σε κάποια άλλη εφαρμογή. Στην τελευταία περίπτωση, έχουμε interprocess communication (IPC). Στην περίπτωση που πρόκειται για την ίδια εφαρμογή, το service object μπορεί να χρησιμοποιηθεί απευθείας από τα άλλα components. Ένα bound service τερματίζεται, όταν και το τελευταίο component αποσυνδέεται από αυτό, εκτός αν είναι επίσης και started.

Οι κυριότερες callback μέθοδοι, στην περίπτωση του service, είναι οι ακόλουθες:

onCreate()	Καλείται κατά τη δημιουργία του service, πριν την κλήση των onStartCommand ή onBind. Δεν καλείται, αν το service εκτελείται ήδη.
onDestroy ()	Καλείται όταν το service δε χρησιμοποιείται πλέον, πριν την καταστροφή του από το σύστημα.
onStartCommand()	Καλείται όταν ένα component μιας εφαρμογής εκκινεί το service, καλώντας τη startService. Μετά την εκτέλεση της onStartCommand το service θεωρείται started.
onBind()	Καλείται όταν ένα component μιας εφαρμογής συνδέεται (bind) με το service, καλώντας την bindService(). Η μέθοδος αυτή επιστρέφει το interface, που ορίζεται από το bound service, για την επικοινωνία του με άλλα components.
onUnbind()	Καλείται όταν και το τελευταίο συνδεμένο component αποσυνδέεται.

Ο κύκλος ζωής ενός service και οι αντίστοιχες μέθοδοι παρουσιάζονται στην εικόνα της διπλανής σελίδας.



Service Lifecycle

5.4.3 Content Provider

Ένας content provider είναι ένα στοιχείο μιας εφαρμογής, που ως σκοπό έχει τη διαχείριση της πρόσβασης σε ένα δομημένο σύνολο δεδομένων. Ορίζει ένα authority και ένα σύνολο από URIs (Uniform Resource Identifiers), μέσω των οποίων, άλλα components μπορούν να αποκτήσουν πρόσβαση στα δεδομένα που ενθυλακώνονται από τον provider. Η πρόσβαση αυτή, μπορεί να είναι, τόσο για ανάκτηση των δεδομένων, όσο και για μεταβολή τους (π.χ. προσθήκη, διαγραφή, ενημέρωση). Ο content provider παρέχει επίσης μηχανισμούς, μέσω των οποίων είναι δυνατόν να οριστεί το απαιτούμενο επίπεδο ασφάλειας των δεδομένων.

Για να αποκτήσει πρόσβαση στα δεδομένα ενός content provider, ένα component πρέπει να χρησιμοποιήσει τον ContentResolver, ένα αντικείμενο που παρέχεται από το σύστημα στις εφαρμογές. Το component περνάει στον ContentResolver τα URIs των δεδομένων που το ενδιαφέρουν. Στη συνέχεια, ο ContentResolver αναζητά τον content provider που έχει δηλωθεί για τα συγκεκριμένα URIs. Αφού τον εντοπίσει, του στέλνει το αίτημα του component. Ο content provider επεξεργάζεται το αίτημα και επιστρέφει τα αποτελέσματα. Κατά τη διαδικασία αυτή, είναι πιθανόν να υπάρχουν έλεγχοι ως προς τα δικαιώματα του component σε σχέση με τα δεδομένα, τα οποία θέλει να ανακτήσει ή να επεξεργαστεί.

Ο μηχανισμός των content providers είναι ο τρόπος με τον οποίο μια εφαρμογή, η οποία αποθηκεύει δεδομένα που ενδιαφέρουν και άλλες εφαρμογές, μπορεί να παρέχει πρόσβαση στα δεδομένα αυτά, με δομημένο και ασφαλή τρόπο. Ένα παράδειγμα content providers είναι ο Contacts Provider, ο οποίος αποτελεί μέρος της πλατφόρμας του Android. Ο provider αυτός ορίζει κατάλληλα URIs, μέσω των οποίων άλλες εφαρμογές, με τα κατάλληλα δικαιώματα, μπορούν να αποκτήσουν πρόσβαση στις επαφές του χρήστη.

5.4.4 Broadcast Receiver

Ένας broadcast receiver είναι ένα στοιχείο μιας εφαρμογής, που ως σκοπό έχει την αποδοχή system-wide broadcast μηνυμάτων, που στέλνονται, είτε από το ίδιο το λειτουργικό σύστημα, είτε από άλλες εφαρμογές. Τα μηνύματα αυτά συνδέονται, συνήθως, με κάποιο γεγονός, όπως, για παράδειγμα, ότι η οθόνη έκλεισε, ότι το επίπεδο της μπαταρίας είναι χαμηλό, ότι κάποιο download που εκτελούνταν στο παρασκήνιο ολοκληρώθηκε. Ένας broadcast receiver δεν έχει user interface. Συνήθως, η λειτουργία του είναι να εκτελεί κάποιες απλές ενέργειες, σχετικές με το γεγονός που συνέβη, ή να ενεργοποιεί άλλα components, προκειμένου να εκτελέσουν πιο σύνθετες ενέργειες.

5.4.5 Intent

Τα intents δεν αποτελούν στοιχεία εφαρμογής, τα περιγράφουμε, όμως, εδώ, επειδή σχετίζονται άμεσα με αυτά. Από τα παραπάνω τέσσερα components, τα τρία (activity, service και broadcast receiver) ενεργοποιούνται μέσω μηνυμάτων, που ονομάζονται intents. Τα intents αποτελούν ένα πολύ βασικό στοιχείο της αρχιτεκτονικής του Android, αφού είναι ο κύριος μηχανισμός αποστολής μηνυμάτων μεταξύ components της ίδιας ή διαφορετικών εφαρμογών. Ένα intent μπορεί να περιέχει δεδομένα σχετικά με μια επιθυμητή ενέργεια ή, όπως στην περίπτωση των broadcasts, την περιγραφή του γεγονότος που συνέβη.

Υπάρχουν δύο είδη intents, που περιγράφουν επιθυμητές ενέργειες, τα explicit και τα implicit. Ένα explicit intent ορίζει επακριβώς το component στο οποίο απευθύνεται, ενώ, ένα implicit intent ορίζει απλώς την ενέργεια, χωρίς συγκεκριμένο παραλήπτη. Ένα παράδειγμα implicit intent είναι όταν μια εφαρμογή θέλει να ανοίξει μια ιστοσελίδα. Αντί να χρειάζεται να έχει η ίδια δυνατότητες αναπαράστασης ιστοσελίδων, μπορεί να χρησιμοποιήσει τον browser της πλατφόρμας. Χρησιμοποιώντας implicit intent, η εφαρμογή δε χρειάζεται να γνωρίζει ποιος είναι ο browser που χρησιμοποιεί ο χρήστης. Απλώς ζητάει από το σύστημα να ανοίξει μια ιστοσελίδα, και είναι το σύστημα αυτό, το οποίο αναζητά την κατάλληλη εφαρμογή, ίσως και με ερώτηση στον χρήστη, αν υπάρχουν περισσότερες από μία.

5.5 Αρχεία XML

5.5.1 Χρήση

Το Android API κάνει ευρεία χρήση αρχείων XML σε πάρα πολλά σημεία του. Μέσω κατάλληλης σύνταξης XML, που ορίζεται για κάθε περίπτωση, είναι δυνατό να οριστούν η μορφή του UI των activities και widgets, strings, integers, menus, γραφικά, χρώματα, μεγέθη, στυλ, οθόνες για ορισμό ρυθμίσεων (preference screens) και, γενικά, τα περισσότερα είδη resources που ορίζονται στο API. Στις περισσότερες περιπτώσεις, μάλιστα, τα αντικείμενα που ορίζονται μέσω XML, έχουν αντίστοιχο αντικείμενο, ορισμένο ως κλάση της Java, με πολλά από τα XML elements και attributes να αντιστοιχούν σε συγκεκριμένες μεταβλητές και μεθόδους.

Ο ορισμός παραμέτρων της εφαρμογής μέσω αρχείων XML και η αποφυγή hard-coded μεγεθών έχει το πλεονέκτημα ότι η παραμετροποίηση μπορεί να γίνει κεντρικά. Έτσι, αλλάζοντας για παράδειγμα ένα χρώμα ορισμένο ως resource μέσω XML, ανανεώνονται όλα τα σημεία της εφαρμογής, που χρησιμοποιούν αυτό το χρώμα. Ειδικότερα για τα strings του user interface, ο ενδεδειγμένος τρόπος ορισμού είναι μέσω XML, ώστε να μπορούν να μεταφραστούν εύκολα, χωρίς να χρειάζεται να αλλάξει ο κώδικας της εφαρμογής.

Επίσης, όπως αναφέραμε κατά την περιγραφή του activity component, το UI, στις Android εφαρμογές, ορίζεται, συνήθως, με τη βοήθεια μιας σειράς από αρχεία XML με κατάλληλη σύνταξη, που έχει οριστεί για τον σκοπό αυτό. Αυτός ο τρόπος ορισμού του UI, εκτός του ότι κάνει τον κώδικα υλοποίησης πιο γενικό, αφού αυτός δεν εξαρτάται πλέον από μια συγκεκριμένη σχεδίαση UI, κάνει τις αλλαγές στο UI ευκολότερες. Μάλιστα, χρησιμοποιώντας διαδικασίες που παρέχονται από το Android API, είναι δυνατόν να οριστούν διαφορετικά αρχεία για διαφορετικού μεγέθους συσκευές ή ακόμα και για διαφορετικό προσανατολισμό της οθόνης. Έτσι, χωρίς να χρειάζονται αλλαγές στον κώδικα των activities, τα UI, που αυτά παρουσιάζουν, μπορούν να μεταβάλλονται ανάλογα με την εκάστοτε χρήση τους.

5.5.2 Android Manifest

Ένα από τα σημαντικότερα αρχεία XML μιας εφαρμογής Android, είναι το AndroidManifest.xml. Αυτό περιέχει όλες τις απαραίτητες πληροφορίες που χρειάζεται το σύστημα, προκειμένου να μπορεί να εκτελέσει μια εφαρμογή, καθώς και πολλά άλλα δεδομένα που σχετίζονται με αυτή. Περιλαμβάνει δεδομένα όπως το όνομα και η έκδοση της εφαρμογής, το εικονίδιο με το οποίο εμφανίζεται στη λίστα των εφαρμογών, η έκδοση του Android (ουσιαστικά το API level) για την οποία έχει γραφτεί, καθώς και το σύνολο των απαιτούμενων χαρακτηριστικών (features) και αδειών (permissions), που απαιτεί για τη λειτουργία της.

Ο ορισμός API level και features δίνει τη δυνατότητα σε υπηρεσίες τύπου Play Store να μην παρουσιάζουν εφαρμογές σε κάποιο χρήστη ή να μην τις εγκαθιστούν, αν αυτές δεν μπορούν να τρέξουν στη συσκευή του. Ο ορισμός των permissions είναι απαραίτητος, αν η εφαρμογή θέλει να έχει πρόσβαση σε υπηρεσίες και δεδομένα του συστήματος, που απαιτούν τα αντίστοιχα permissions. Το Android ορίζει μια σειρά από permissions, τα οποία απαιτούνται προκειμένου να εκτελεστούν συγκεκριμένες λειτουργίες του συστήματος. Για παράδειγμα, μια εφαρμογή χρειάζεται το κατάλληλο permission για να διαβάσει τις επαφές, να ανοίξει την κάμερα, να αποκτήσει πρόσβαση στο Internet, να γράψει στην εξωτερική μνήμη της συσκευής ή, όπως στην περίπτωση της δικής μας εφαρμογής, να χρησιμοποιήσει το Bluetooth. Κατά την εγκατάσταση μιας εφαρμογής ζητείται από τον χρήστη, αφού

εξετάσει ποιες άδειες ζητά η συγκεκριμένη εφαρμογή, να δώσει τη συγκατάθεσή του. Διαφορετικά, η εφαρμογή δεν εγκαθίσταται.

Ένα άλλο πολύ σημαντικό τμήμα του Android manifest είναι η δήλωση των components της εφαρμογής. Εκτός από τους broadcast receivers, οι οποίοι μπορούν να δηλωθούν και προγραμματιστικά, όλα τα υπόλοιπα components δεν είναι δυνατόν να χρησιμοποιηθούν, αν πρώτα δε έχουν δηλωθεί στο manifest. Μέσω του manifest, το σύστημα μπορεί να βρει το συγκεκριμένο component, που πρέπει να ενεργοποιηθεί ως απάντηση σε κάποιο intent. Στο προηγούμενο παράδειγμα, μια εφαρμογή browser δηλώνει στο manifest αρχείο της, τη δυνατότητά της να απαντά στο συγκεκριμένο implicit intent με την εκκίνηση του αντίστοιχου activity.

5.6 Λοιπά στοιχεία

5.6.1 SQLite Database

Η SQLite είναι μία μηχανή βάσης δεδομένων, ειδικά σχεδιασμένη για ενσωματωμένα συστήματα. Δεν έχει ξεχωριστή διεργασία server, αλλά, αντίθετα, εκτελείται μέσα στη διεργασία της εφαρμογής που τη χρησιμοποιεί. Η πλήρης βάση δεδομένων είναι αποθηκευμένη σε ένα μόνο αρχείο. Το Android παρέχει κατάλληλο API για τη δημιουργία βάσεων δεδομένων, χρησιμοποιώντας την SQLite, καθώς και API για την εκτέλεση ερωτημάτων και εντολών SQL πάνω σε αυτές.

Οι περισσότερες μηχανές βάσης δεδομένων χρησιμοποιούν στατικούς τύπους για τα δεδομένα που αποθηκεύουν. Η SQLite, αντίθετα, χρησιμοποιεί ένα πιο γενικό σύστημα δυναμικών τύπων. Ο τύπος ενός αντικειμένου στην SQLite δεν εξαρτάται από τη στήλη της βάσης δεδομένων στην οποία βρίσκεται, αλλά είναι συσχετισμένος με το ίδιο το αντικείμενο. Μπορούν δηλαδή αντικείμενα που βρίσκονται στην ίδια στήλη να έχουν διαφορετικούς τύπους. Η SQLite ορίζει πέντε Storage Classes για τα δεδομένα της:

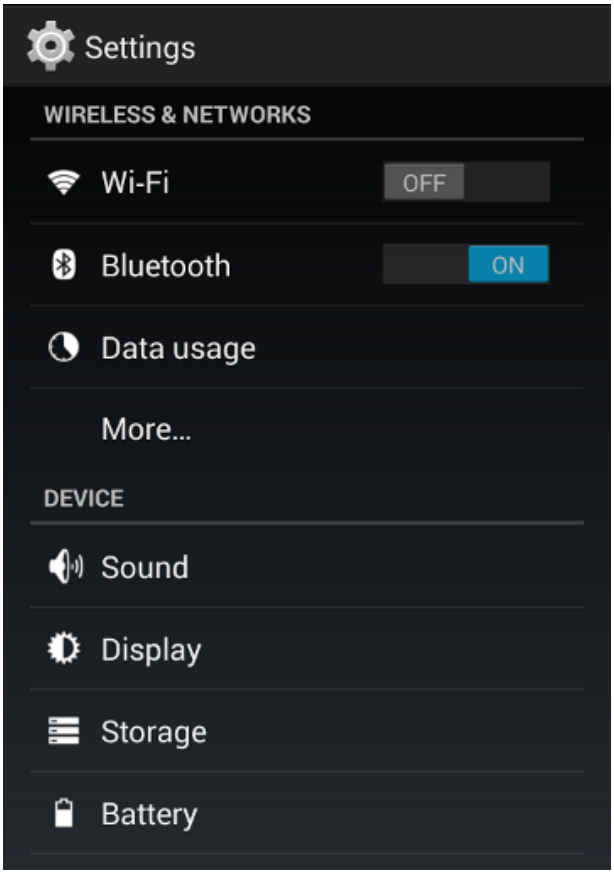
- NULL
- INTEGER: signed integer (1, 2, 3, 4, 6, ή 8 bytes)
- REAL: floating point (8-byte IEEE floating point number)
- TEXT: text string (no data length)
- BLOB: data blob (no data length)

Με τη βοήθεια αυτών μπορούν να αποθηκευτούν και οι υπόλοιποι τύποι, που ορίζονται στην SQL, αλλά δεν υπάρχουν στην SQLite. Για παράδειγμα μια ημερομηνία μπορεί να αποθηκευτεί ως ένας ακέραιος αριθμός (π.χ. seconds since 1-1-1970). Η SQLite έχει, βέβαια, και τους περιορισμούς της. Ένας από τους βασικότερους περιορισμούς, που είχε παλαιότερα, ήταν ότι δεν επέβαλε τους κανόνες που ορίζονται στην SQL για τα foreign keys. Για λόγους προς τα πίσω συμβατότητας, παρότι πλέον υποστηρίζει foreign keys, αυτά πρέπει να ενεργοποιηθούν, τόσο στη βιβλιοθήκη κατά το compilation, όσο και στο runtime από την εκάστοτε εφαρμογή.

5.6.2 Preferences

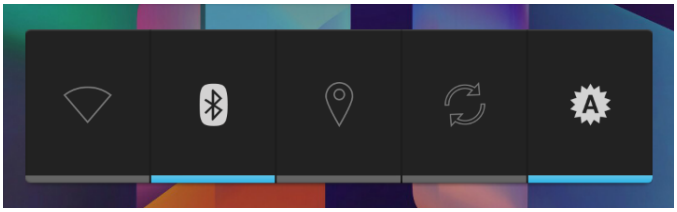
Το Android δίνει τη δυνατότητα στις εφαρμογές να αποθηκεύουν τα δεδομένα τους και τις επιλογές του χρήστη, μέσω του SharedPreferences API. Τα δεδομένα αυτά

αποθηκεύονται σε XML αρχεία στον τοπικό φάκελο της εφαρμογής. Παράλληλα, δίνει τη δυνατότητα στους developers να δημιουργήσουν UI, για τη ρύθμιση παραμέτρων από τον χρήστη, ορίζοντάς τα με κατάλληλα XML αρχεία. Τα συγκεκριμένα UI αποθηκεύουν αυτόματα τις επιλογές του χρήστη σε αρχεία SharedPreferences, ενώ η εφαρμογή έχει τη δυνατότητα να ειδοποιείται όταν αλλάζει κάποιο preference. Το πλεονέκτημα της μεθόδου αυτής, εκτός από το γεγονός ότι απαιτείται λιγότερος κώδικας, είναι ότι το interface που παράγεται, είναι παρόμοιο με αυτό, που το ίδιο το σύστημα χρησιμοποιεί για τον ορισμό επιλογών από τον χρήστη. Έτσι, η εφαρμογή εμφανίζει στον χρήστη ένα UI ρυθμίσεων, το οποίο είναι συνεπές με την πλατφόρμα και την έκδοση στην οποία τρέχει, οπότε ο χρήστης μπορεί να εξοικειωθεί άμεσα με αυτό.



5.6.3 App Widgets

Τα app widgets είναι μικρά σε μέγεθος και απλά user interfaces ή views, τα οποία μπορούν να ενσωματωθούν από μία εφαρμογή σε μία άλλη. Η δεύτερη περιέχει ένα component που ονομάζεται App Widget Host. Τα app widgets μπορούν να χρησιμοποιηθούν, ώστε να παρέχουν συνοπτικές πληροφορίες στον χρήστη για τη λειτουργία της εφαρμογής ή για δεδομένα που μπορεί να τον ενδιαφέρουν, τα οποία είναι συνήθως μεταβαλλόμενα. Επίσης, ένα app widget έχει τη δυνατότητα να απαντά σε απλές ενέργειες του χρήστη (π.χ. click πάνω στο app widget για άνοιγμα της εφαρμογής), ενώ, ανά καθορισμένα χρονικά διαστήματα, του ζητείται, από τον App Widget Host, να ανανεώσει τα περιεχόμενά του. Το πιο κλασικό παράδειγμα App Widget Host είναι η εφαρμογή Home Screen της συσκευής, στην οποία ο χρήστης μπορεί να προσθέσει, να αφαιρέσει ή να μετακινήσει app widgets, που προσφέρονται από τις εγκατεστημένες εφαρμογές.



Ένα app widget είναι, στην ουσία, ένας broadcast receiver, ο οποίος μπορεί να δέχεται συγκεκριμένα μηνύματα (intents) που έχουν οριστεί στο Android API. Για να μπορεί το σύστημα να βρει το app widget και να το παρουσιάσει ως επιλογή στον χρήστη, πρέπει αυτό να οριστεί στο Android Manifest ως broadcast receiver, με κάποια επιπλέον μετα-δεδομένα. Σε αυτά περιέχεται και ένα αρχείο XML, στο οποίο ορίζονται βασικές παράμετροι του app widget, όπως το επιθυμητό μέγεθος, η δυνατότητα αλλαγής μεγέθους, το αρχικό interface που παρουσιάζει στον χρήστη και η περίοδος των αιτημάτων ανανέωσης από τον App Widget Host. Μπορεί, επίσης, να οριστεί ένα activity, το οποίο δίνει τη δυνατότητα στον χρήστη να καθορίζει παραμέτρους του app widget. Έτσι, μπορεί αυτό να προστεθεί περισσότερες από μία φορές στον App Widget Host, με κάθε εκδοχή του να εμφανίζει διαφορετικά δεδομένα.

5.7 Android Bluetooth Low Energy API

Η επίσημη υποστήριξη του Bluetooth Low Energy στο Android ξεκίνησε σχετικά πρόσφατα με την έκδοση 4.3 και το API Level 18. Κάποιοι κατασκευαστές κινητών, βέβαια, προσέφεραν ήδη υποστήριξη στις συσκευές τους, καθώς και δικά τους API, ώστε να μπορούν οι developers να δημιουργούν εφαρμογές BLE. Όπως ισχύει και για άλλες πλατφόρμες κινητών συσκευών, μόνο οι τελευταίας γενιάς συσκευές υποστηρίζουν το BLE. Αυτό, βέβαια, είναι αναμενόμενο, αφού πρόκειται για νέα τεχνολογία, και οι παλαιότερες συσκευές δεν έχουν το κατάλληλο Bluetooth hardware, ώστε να παρέχουν υποστήριξη (δεν αρκεί, δηλαδή, ένα software update).

5.7.1 Βασικά Στοιχεία

Προκειμένου μια εφαρμογή να χρησιμοποιήσει το Bluetooth, πρέπει πρώτα να δηλώσει τα αντίστοιχα permissions στο Android Manifest. Για το Bluetooth, αυτά είναι τα BLUETOOTH και BLUETOOTH_ADMIN.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Επίσης, η εφαρμογή μπορεί να δηλώσει ότι το BLE είναι απαραίτητο για τη λειτουργία της, ώστε να αποφεύγονται περιπτώσεις κατά τις οποίες ο χρήστης εγκαθιστά την εφαρμογή σε μια συσκευή, η οποία δεν παρέχει υποστήριξη, οπότε η εφαρμογή δε λειτουργεί.

```
<uses-feature
    android:name="android.hardware.bluetooth_le"
    android:required="true" />
```

Η πρόσβαση στο Bluetooth hardware γίνεται μέσω του Bluetooth Manager, ο οποίος μπορεί να επιστρέψει ένα αντικείμενο BluetoothAdapter. Το αντικείμενο αυτό αντιπροσωπεύει το Bluetooth hardware της συσκευής και μπορεί να εκτελέσει όλες τις

αντίστοιχες λειτουργίες, όπως για παράδειγμα η αναζήτηση συσκευών. Άλλο σημαντικό αντικείμενο του API είναι το BluetoothDevice, το οποίο αντιπροσωπεύει μία συσκευή Bluetooth που έχει βρεθεί, και παρέχει δυνατότητες όπως σύνδεση και bonding.

```
BluetoothManager manager = (BluetoothManager)
            context.getSystemService(Context.BLUETOOTH_SERVICE);
if (manager == null) return;
BluetoothAdapter adapter = manager.getAdapter();
```

Αν το Bluetooth είναι απενεργοποιημένο, η εφαρμογή έχει τη δυνατότητα να ζητήσει από το σύστημα την ενεργοποίησή του, με τη συγκατάθεση του χρήστη.

```
if (!adapter.isEnabled()) {
    Intent enable = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    activity.startActivityForResult(enable, requestCode);
}
```

Μπορεί επίσης να ελέγξει και στο runtime, την ύπαρξη BLE hardware.

```
if (!getPackageManager().hasSystemFeature(
        PackageManager.FEATURE_BLUETOOTH_LE)) {
    // BLE not available
}
```

Τέλος, μπορεί να κάνει register ένα broadcast receiver, προκειμένου να δέχεται μηνύματα σχετικά με αλλαγές στην κατάσταση του Bluetooth, όπως ενεργοποίηση ή απενεργοποίηση.

5.7.2 Περιγραφή του API

Το BLE API του Android είναι χτισμένο πάνω στο GATT. Πολλές από τις λειτουργίες που παρέχει έχουν άμεση σχέση με τις αντίστοιχες λειτουργίες του GATT, τις οποίες είδαμε σε προηγούμενο κεφάλαιο. Το API παρέχει μια σειρά από interfaces και callbacks, μέσω των οποίων μια εφαρμογή μπορεί να ανακαλύψει και να χρησιμοποιήσει τις GATT υπηρεσίες μιας BLE συσκευής. Είναι περισσότερο προσανατολισμένο στη λειτουργία ως GATT client και όχι ως server. Αυτό δεν αποτελεί πρόβλημα, αφού στις περισσότερες περιπτώσεις χρήσης, μια Bluetooth Smart Ready συσκευή, όπως ένα Android smartphone, συνδέεται ως GATT client με μια Bluetooth Smart συσκευή, που λειτουργεί ως GATT server. Εδώ πρέπει να πούμε, ότι, επειδή έχουμε να κάνουμε με πρωτόκολλα δικτύου, οι περισσότερες λειτουργίες εκτελούνται ασύγχρονα. Δηλαδή, η εφαρμογή εκκινεί μια διαδικασία, και, αργότερα, ενημερώνεται για τα αποτελέσματα μέσω μιας callback μεθόδου.

Ακολουθεί συνοπτική περιγραφή της χρήσης του BLE API από μια εφαρμογή. Η εφαρμογή, λοιπόν, αφού ελέγξει ότι όλα τα απαραίτητα στοιχεία λειτουργούν όπως πρέπει, ξεκινά την αναζήτηση BLE συσκευών με τη μέθοδο startLeScan του BluetoothAdapter. Αυτή δέχεται ως όρισμα το interface BluetoothAdapter.LeScanCallback, το οποίο περιέχει την callback μέθοδο onLeScan, η οποία και καλείται, για κάθε BLE συσκευή που βρίσκεται από την αναζήτηση. Σε αυτό το σημείο, η εφαρμογή μπορεί να ελέγξει τα advertising και scan response data, ώστε να δει αν η συγκεκριμένη συσκευή είναι αυτή την οποία αναζητά. Στην περίπτωση που ισχύει κάτι τέτοιο, μπορεί, στη συνέχεια, να συνδεθεί με αυτή, χρησιμοποιώντας τη μέθοδο connectGatt του BluetoothDevice.

Η μέθοδος connectGatt επιστρέφει ένα αντικείμενο BluetoothGatt. Το συγκεκριμένο αντικείμενο παρέχει κατάλληλες μεθόδους για την εκκίνηση όλων των απαραίτητων για την εφαρμογή GATT sub-procedures: service discovery, characteristic read/write/reliable write,

descriptor read/write. Πριν κάνει οτιδήποτε άλλο, η εφαρμογή πρέπει να καλέσει τη μέθοδο discoverServices, προκειμένου να γίνει η ανακάλυψη των services, characteristics και descriptors του GATT server στην απομακρυσμένη συσκευή. Μετά το service discovery, οι πληροφορίες για τα παραπάνω είναι διαθέσιμες τοπικά, σε αντικείμενα BluetoothGattService, BluetoothGattCharacteristic και BluetoothGattDescriptor. Μετά την ολοκλήρωση ενός read sub-procedures για characteristic ή descriptor, ή αφού γίνει κάποιο notification/indication ενός characteristic, η τιμή του characteristic ή του descriptor είναι διαθέσιμη τοπικά στο αντικείμενο που το αντιπροσωπεύει. Πριν την εκκίνηση ενός write sub-procedure για characteristic ή descriptor, η νέα τιμή πρέπει επίσης να γραφτεί τοπικά. Τέλος, εκτός από τον client characteristic configuration descriptor στον GATT server, που ενεργοποιεί ή απενεργοποιεί τα notifications/indications, το Android API δίνει τη δυνατότητα και τοπικής ρύθμισης, που είναι από default απενεργοποιημένη. Αυτό σημαίνει ότι, προκειμένου η εφαρμογή να δέχεται notifications/indications για characteristics, θα πρέπει πρώτα να τα ενεργοποιήσει, εκτός από απομακρυσμένα, και τοπικά.

Επίσης, η μέθοδος connectGatt δέχεται ως όρισμα ένα αντικείμενο τύπου BluetoothGattCallback, το οποίο παρέχει μια σειρά από callback μεθόδους για διάφορες λειτουργίες του BLE, οι περισσότερες από τις οποίες εκκινούνται μέσω του BluetoothGatt. Οι μέθοδοι είναι οι εξής:

onConnectionStateChange()	Καλείται όταν η συσκευή συνδέεται με ή αποσυνδέεται από την απομακρυσμένη συσκευή BLE. Η αποσύνδεση μπορεί να οφείλεται σε διακοπή της σύνδεσης. BluetoothGatt methods: connect, disconnect
onServicesDiscovered()	Καλείται όταν τα services, τα characteristics και οι descriptors της απομακρυσμένης συσκευής BLE έχουν ανακαλυφθεί. GATT sub-procedures: Discover All Primary Services Find Included Services Discover All Characteristic of a Service Discover All Characteristic Descriptors BluetoothGatt method: discoverServices
onCharacteristicChanged()	Καλείται όταν ένα χαρακτηριστικό γίνεται notify ή indicate από την απομακρυσμένη συσκευή BLE (GATT Notifications και Indications sub-procedures).
onCharacteristicRead()	Καλείται όταν ολοκληρωθεί (επιτυχία ή αποτυχία) ένα GATT Read (Long) Characteristic Value sub-procedure. BluetoothGatt method: readCharacteristic
onCharacteristicWrite()	Καλείται όταν ολοκληρωθεί (επιτυχία ή αποτυχία) ένα από τα sub-procedures του GATT Characteristic Value Write feature. Η επιλογή του τύπου εγγραφής (Write, Write Without Response, Signed Write) γίνεται πριν την εκκίνηση της διαδικασίας. BluetoothGatt method: writeCharacteristic
onDescriptorRead()	Καλείται όταν ολοκληρωθεί (επιτυχία ή αποτυχία) ένα GATT Read (Long) Characteristic Descriptor sub-procedure. BluetoothGatt method: readDescriptor
onDescriptorWrite()	Καλείται όταν ολοκληρωθεί (επιτυχία ή αποτυχία) ένα GATT Write (Long) Characteristic Descriptor sub-procedure. BluetoothGatt method: writeDescriptor
onReliableWriteCompleted()	Καλείται όταν ολοκληρωθεί (επιτυχία ή αποτυχία) ένα GATT Characteristic Value Reliable Write sub-procedure. BluetoothGatt method: executeReliableWrite

onReadRemoteRssi()	Καλείται προκειμένου να επιστρέψει στην εφαρμογή το RSSI της σύνδεσης με την απομακρυσμένη συσκευή BLE. Το RSSI (Received Signal Strength Indication) είναι μέτρηση της ισχύος του λαμβανόμενου σήματος. Η συγκεκριμένη λειτουργία εκτελείται τοπικά στη συσκευή και δεν έχει άμεση σχέση με το BLE, είναι όμως σημαντική σε proximity applications, αφού μπορεί να χρησιμοποιηθεί για τον κατά προσέγγιση προσδιορισμό της απόστασης μεταξύ των συσκευών. BluetoothGatt method: readRemoteRssi

5.7.3 Περιορισμοί

Πρώτα από όλα, πρέπει να αναφέρουμε ότι το BLE API του Android είναι σχετικά καινούριο και, ως εκ τούτου, μπορεί να περιέχει bugs και να παρουσιάζει ενίοτε δυσλειτουργίες και μη αναμενόμενη συμπεριφορά. Αναφέρουμε εδώ κάποιους από τους περιορισμούς που συναντήσαμε.

Όπως είδαμε παραπάνω, προκειμένου να λειτουργήσουν τα notifications και indications πρέπει να ενεργοποιηθούν και τοπικά. Όμως, το API θέτει ένα αυθαίρετο όριο στον αριθμό των notifications/indications που μπορούν να είναι ενεργοποιημένα ταυτόχρονα. Το όριο αυτό ήταν 4 στο API level 18 και αυξήθηκε σε 7 στο API level 19. Αυτό πρέπει να ληφθεί υπόψη, ειδικά σε εφαρμογές σαν τη δική μας, που έχουν να κάνουν με αισθητήρες, αφού η ασύγχρονη αποστολή των αποτελεσμάτων μετρήσεων μέσω notifications, είναι από τις βασικότερες λειτουργίες του συστήματος. Θα δούμε σε επόμενο κεφάλαιο πώς αντιμετωπίσαμε τον συγκεκριμένο περιορισμό.

Ένας άλλος περιορισμός του API είναι ότι δεν περιέχει κάποιου τύπου ουρά για τις λειτουργίες που εκκινούνται από την εφαρμογή. Όπως είδαμε σε προηγούμενο κεφάλαιο, μόνο μία λειτουργία, που απαιτεί απάντηση από τον server, είναι δυνατόν να είναι ενεργή κάθε στιγμή. Άρα, αν υπάρχουν πολλές λειτουργίες προς εκτέλεση, αυτές πρέπει να μπουν σε μια ουρά και να εκτελεστούν σειριακά. Το BLE API του Android δεν προσφέρει τέτοια λειτουργία. Μάλιστα, η εκκίνηση μιας νέας διαδικασίας, ενώ η προηγούμενη δεν έχει ακόμη ολοκληρωθεί, δεν αποτυγχάνει, αλλά απλώς αγνοείται. Θα πρέπει η ίδια η εφαρμογή να αναλάβει τη διαχείριση των BLE διαδικασιών που απαιτεί, υλοποιώντας την απαραίτητη ουρά.

Τέλος, επειδή το BLE API του Android, όπως, βέβαια, και αντίστοιχα APIs για άλλες πλατφόρμες, είναι αρκετά high-level, δεν παρέχει μεγάλες δυνατότητες ελέγχου του BLE hardware (controller). Αυτό έχει βέβαια το πλεονέκτημα ότι καθιστά το API ευκολότερο, αλλά δεν επιτρέπει στους developers να θέτουν τις παραμέτρους λειτουργίας του BLE, όπως μπορεί να γίνει για παράδειγμα στο DA14580. Ένα παράδειγμα περιορισμού που προκύπτει από αυτό, είναι στη διαδικασία του service discovery, ότι αυτή μπορεί να γίνει μόνο συνολικά. Δεν υπάρχει δηλαδή τρόπος να εκτελεστεί μια διαδικασία τύπου Discover Primary Services By Service UUID, που θα απαιτούσε λιγότερη επικοινωνία μεταξύ των συσκευών. Άλλο παράδειγμα είναι η διαδικασία του scanning. Το πρότυπο του BLE, ορίζει τα μεγέθη scan interval και scan window, τα οποία καθορίζουν το scan duty cycle. Στο Android, μια εφαρμογή ξεκινά το scan μέσω του BluetoothAdapter, και αυτό συνεχίζεται (κατά πάσα πιθανότητα με 100% duty cycle) μέχρι η εφαρμογή να το σταματήσει. Αν η εφαρμογή απαιτεί περιοδικό scanning, θα πρέπει να το υλοποιήσει η ίδια.

Κεφάλαιο 6

Weather Station Profile

Σε αυτό το κεφάλαιο, περιγράφουμε τον τρόπο λειτουργίας του συστήματος καταγραφής καιρού. Δείχνουμε, δηλαδή, ποια είναι τα δεδομένα που συλλέγονται από το σύστημα, ποιες οι δυνατές ρυθμίσεις που μπορούν να γίνουν σχετικά με τη λειτουργία του, και πώς, χρησιμοποιώντας το Bluetooth Low Energy, μπορούν τα δύο τμήματα του συστήματος να επικοινωνήσουν και να ανταλλάξουν τα δεδομένα αυτά.

Ουσιαστικά, περιγράφουμε το GATT-profile που δημιουργήσαμε για τη συγκεκριμένη περίπτωση χρήσης, δηλαδή το πρότυπο, το οποίο πρέπει να υλοποιούν οι συμβατές συσκευές, προκειμένου να μπορούν να πραγματοποιήσουν τις λειτουργίες του συστήματος. Για τον σκοπό αυτό, κάνουμε, αρχικά, μια γενική περιγραφή του προφίλ, των ρόλων που ορίζονται σε αυτό και των λειτουργιών που προβλέπονται. Στη συνέχεια, ορίζουμε τα απαραίτητα δεδομένα, τη μορφή τους και τις δυνατές τιμές τους, ενώ προχωράμε στον τυπικό, πλέον, ορισμό του προφίλ μέσω XML. Παρουσιάζουμε τα χαρακτηριστικά τα οποία δημιουργήσαμε για την αποθήκευση των τιμών των δεδομένων και τις υπηρεσίες στις οποίες αυτά περιέχονται. Ακολούθως, περιγράφουμε πώς οι υπηρεσίες αυτές συνδυάζονται στους ρόλους που προβλέπονται από το προφίλ, ώστε να υλοποιείται η συγκεκριμένη περίπτωση χρήσης. Τέλος, περιγράφουμε την αναμενόμενη συμπεριφορά κάθε ρόλου σε συγκεκριμένες λειτουργίες του προφίλ και του BLE.

6.1 Καιρικά δεδομένα

Στην υπο-ενότητα αυτή, αναφέρουμε τα βασικά μεγέθη, τα οποία έχουν σχέση με τον καιρό, και μπορούν να συμπεριλαμβάνονται στις μετρήσεις του συστήματος.

6.1.1 Θερμοκρασία – Υγρασία

Τα βασικότερα, σχετικά με τον καιρό, μεγέθη είναι η θερμοκρασία, η υγρασία και η ατμοσφαιρική (βαρομετρική) πίεση. Η θερμοκρασία και η υγρασία δίνουν άμεσες ενδείξεις για τον καιρό που επικρατεί και την τρέχουσα κατάσταση του περιβάλλοντος. Επίσης με το συνδυασμό τους, μπορούν να υπολογιστούν μεγέθη όπως το Heat Index ή το Humidex, τα οποία αποτελούν ένα μέτρο του πώς ένας άνθρωπος αισθάνεται τη θερμοκρασία υπό τις συγκεκριμένες συνθήκες. Μιλάμε, δηλαδή, για έννοιες όπως real feel temperature, felt air temperature, apparent temperature, που χρησιμοποιούνται για να εκφράσουν τα συγκεκριμένα μεγέθη. Στην περίπτωση της υγρασίας, το μέγεθος που μετρείται, συνήθως, είναι η σχετική υγρασία (relative humidity – RH). Το RH είναι ο λόγος της τρέχουσας ποσότητας υδρατμών στον αέρα και της τρέχουσας μέγιστης δυνατής ποσότητας υδρατμών, που μπορεί να περιέχεται σε αυτόν. Εξαρτάται από τη θερμοκρασία και την πίεση του αέρα. Δίνεται με τη μορφή ποσοστού τοις εκατό και δεν έχει μονάδα μέτρησης.

6.1.2 Ατμοσφαιρική Πίεση

Η ατμοσφαιρική πίεση, εκτός και αν είναι πολύ υψηλή ή χαμηλή, δε δίνει άμεσες ενδείξεις για τον καιρό. Είναι, όμως, ένα από τα σημαντικότερα μεγέθη σχετικά με την πρόγνωση του καιρού. Με τον έλεγχο των τιμών της, τόσο σε τοπικό όσο και σε ευρύτερο επίπεδο, μπορεί να διαπιστωθεί η τάση που παρουσιάζει το συγκεκριμένο μέγεθος, αν δηλαδή έχει ανοδικές ή καθοδικές τάσεις ή παραμένει σταθερό. Το τελευταίο αποτελεί μία από τις κύριες μετρικές, πάνω στις οποίες βασίζονται τα συστήματα πρόγνωσης του καιρού. Σε αυτή την περίπτωση έχουμε έννοιες όπως συστήματα υψηλής και χαμηλής πίεσης, περιοχές δηλαδή της ατμόσφαιρας στις οποίες η ατμοσφαιρική πίεση παρουσιάζει τοπικό μέγιστο ή

ελάχιστο. Αυτές μπορούν να χρησιμοποιηθούν στην κατάρτιση χαρτών καιρού, όπου εμφανίζονται μέσω ισοβαρών (γραμμών που αντιστοιχούν σε περιοχές με όμοια βαρομετρική πίεση). Οι χάρτες καιρού χρησιμοποιούνται για την έρευνα και πρόγνωση του καιρού.

Σε γενικές γραμμές, η ανοδική τάση της ατμοσφαιρικής πίεσης σημαίνει συνήθως βελτίωση του καιρού, ενώ η καθοδική τάση σημαίνει επιδείνωση (συννεφιά ή/και βροχή). Εδώ παίζει ρόλο και ο ρυθμός με τον οποίο μεταβάλλεται η τιμή της πίεσης. Για παράδειγμα, η απότομη πτώση της ατμοσφαιρικής πίεσης σε μια περιοχή αποτελεί ένδειξη επερχόμενης καταιγίδας.

Η μονάδα μέτρησης της ατμοσφαιρικής πίεσης στο SI είναι το Pascal (1 Pa = 1 Nt/m^2). Στην πράξη χρησιμοποιείται συνήθως ένα πολλαπλάσιο του Pascal, το hPa (100 Pa), το οποίο ονομάζεται επίσης και millibar. Το τελευταίο είναι υποπολλαπλάσιο του bar (100000 Pa). Επίσης, έχουμε την ατμόσφαιρα (ως μονάδα μέτρησης), η οποία αντιστοιχεί στη συνηθισμένη τιμή της πίεσης στο επίπεδο της θάλασσας και είναι ίση με 101325 Pa. Τέλος, έχουμε το psi (6894.76 Pa), το mmHg (Torr) (133.32 Pa) και το inHg (3386.39 Pa).

Η τιμή της ατμοσφαιρικής πίεσης κυμαίνεται συνήθως μεταξύ 980 hPa και 1050 hPa. Οι τιμές αυτές αφορούν την ατμοσφαιρική πίεση στο επίπεδο της θάλασσας, αφού η τιμή της εξαρτάται σε μεγάλο βαθμό από το υψόμετρο.

6.1.3 Λοιπά μεγέθη

Άλλα μεγέθη σχετικά με τον καιρό είναι η ταχύτητα και η διεύθυνση του ανέμου, καθώς και η ποσότητα της βροχής που έχει πέσει σε μια περιοχή. Η ταχύτητα του ανέμου μπορεί να χρησιμοποιηθεί για τον υπολογισμό του Wind Chill, ένα μέγεθος αντίστοιχο του Heat Index, αλλά για χαμηλές θερμοκρασίες. Η ποσότητα της βροχής που έχει πέσει, ενώ είναι δυνατόν να δώσει ένδειξη για την τρέχουσα κατάσταση του καιρού (π.χ. αν βρέχει εκείνη τη στιγμή), χρησιμοποιείται συνήθως για τη δημιουργία ιστορικών στατιστικών στοιχείων σχετικά με τον καιρό μιας περιοχής.

6.1.4 Ambient Light – UV Index

Τα μεγέθη Ambient Light και UV Index δε σχετίζονται άμεσα με τον καιρό, όμως μπορούν να χρησιμοποιηθούν, προκειμένου να παρέχουν σε ένα χρήστη πληροφορίες σχετικά με την κατάσταση του περιβάλλοντος.

Το Ambient Light μετρείται σε lux και μπορεί να δώσει ενδείξεις για την ποσότητα του ηλιακού φωτός σε μια περιοχή, από την οποία μπορούν να εξαχθούν συμπεράσματα σχετικά με το αν υπάρχει πλήρης ηλιοφάνεια, μερική ή πλήρης κάλυψη του ήλιου από σύννεφα. Ακολουθούν ενδεικτικές τιμές του μεγέθους για διάφορες περιπτώσεις:

Ambient Light	**Κατάσταση περιβάλλοντος**
> 100,000 lux	Πλήρης ηλιοφάνεια
1000 – 2000 lux	Τυπική συννεφιά (μεσημέρι)
< 200 lux	Συννεφιά καταιγίδας (μαύρα σύννεφα, μεσημέρι)
400 lux	Ηλιοφάνεια (ανατολή, δύση)
40 lux	Συννεφιά (ανατολή, δύση)
< 1 lux	Νύχτα

Το UV Index είναι ένα μέγεθος, με το οποίο μετρείται η ισχύς της υπεριώδους ακτινοβολίας του ήλιου σε μια περιοχή. Η τιμή αυτή υπολογίζεται με βάση την ισχύ της ακτινοβολίας σε διαφορετικές περιοχές του υπεριώδους φάσματος, χρησιμοποιώντας κατάλληλους συντελεστές. Με αυτόν τον τρόπο παράγεται μία γραμμική κλίμακα, με τιμές που σχετίζονται άμεσα με το βαθμό επικινδυνότητας για τον άνθρωπο, από την έκθεσή του στην υπεριώδη ακτινοβολία. Το χαρακτηριστικό της γραμμικότητας σημαίνει ότι διπλάσια τιμή του UV Index αντιστοιχεί σε διπλάσια ακτινοβολία. Οπότε, μπορούμε επίσης να θεωρήσουμε ότι διπλάσια τιμή του UV Index αντιστοιχεί σε διπλάσιο κίνδυνο και ο συνιστώμενος μέγιστος χρόνος έκθεσης μειώνεται στο μισό.

Οι διάφορες τιμές του UV Index χρησιμοποιούνται για τη δημιουργία μιας αριθμητικής και χρωματικής κλίμακας, με βάση το βαθμό επικινδυνότητας. Η χρησιμότητά της είναι να βοηθήσει τους ανθρώπους να πάρουν κατάλληλα μέτρα για την αποτελεσματική προστασία τους από την υπεριώδη ακτινοβολία, η οποία, σε βραχυπρόθεσμο στάδιο, μπορεί να προκαλέσει εγκαύματα ή βλάβες στα μάτια, ενώ σε μακροπρόθεσμο καταρράκτη και καρκίνο του δέρματος. Οι τιμές και οι αντίστοιχοι βαθμοί επικινδυνότητας φαίνονται στον πίνακα που ακολουθεί.

UV Index	Βαθμός επικινδυνότητας
0 – 2.9	Μικρός κίνδυνος
3 – 5.9	Μέσος κίνδυνος
6 – 7.9	Μεγάλος κίνδυνος
8 – 10.9	Πολύ μεγάλος κίνδυνος
> 11	Μέγιστος κίνδυνος

Οι υπεύθυνοι οργανισμοί υγείας των διαφόρων χωρών εκδίδουν οδηγίες για τα μέτρα προστασίας που πρέπει να ληφθούν σε κάθε περίπτωση.

6.2 Γενική περιγραφή του προφίλ

Σε αυτή την υπο-ενότητα αναφέρουμε συνοπτικά τα συστατικά στοιχεία του προφίλ που υλοποιήσαμε και τις σχεδιαστικές επιλογές που έγιναν κατά τη δημιουργία του. Τα επιμέρους στοιχεία θα περιγραφούν με λεπτομέρεια παρακάτω.

Όπως είδαμε σε προηγούμενο κεφάλαιο, το GATT λειτουργεί με τρόπο βασισμένο στις υπηρεσίες. Αυτές περιέχουν τα δεδομένα της εφαρμογής, μέσα σε χαρακτηριστικά, και ορίζουν συγκεκριμένες συμπεριφορές για αυτά. Ένας GATT client μπορεί να χρησιμοποιήσει τις υπηρεσίες, τις οποίες παρέχει ένας GATT server, ακόμα και αν δεν αναγνωρίζει όλες τις υπηρεσίες του server, με την προϋπόθεση ότι οι υπηρεσίες λειτουργούν και στις δυο συσκευές όπως ορίζεται στο αντίστοιχο πρότυπο.

Με βάση το τελευταίο, αποφασίσαμε να ορίσουμε μια ξεχωριστή υπηρεσία για το κάθε μέγεθος το οποίο μετρείται από το σύστημα καταγραφής καιρού, στην προκειμένη περίπτωση την ενσωματωμένη εφαρμογή, που εκτελείται στο DA14580, και χρησιμοποιεί μία σειρά από αισθητήρες μέτρησης καιρικών δεδομένων. Έτσι, μια συσκευή, ακόμα και αν δεν αναγνωρίζει όλες τις υπόλοιπες υπηρεσίες, θα μπορούσε για παράδειγμα να λαμβάνει

μετρήσεις θερμοκρασίας από το σύστημα, χρησιμοποιώντας μόνο την υπηρεσία που αντιστοιχεί στη συγκεκριμένη μέτρηση.

Ορίσαμε, λοιπόν, από μία υπηρεσία για κάθε αισθητήρα που περιέχεται στο σύστημα και, επιπλέον. μια γενική υπηρεσία, για λόγους που θα εξηγήσουμε κατά την περιγραφή της. Στη συνέχεια, με τη βοήθεια των υπηρεσιών αυτών, ορίσαμε ένα προφίλ, το Weather Station Profile. Το Weather Station Profile ορίζει δύο ρόλους:

- **Weather Station Role**: Είναι ένας GAP peripheral GATT server. Αποτελεί το τμήμα του συστήματος (η συσκευή καταγραφής), που καταγράφει τα καιρικά δεδομένα, χρησιμοποιώντας μια σειρά από αισθητήρες. Στην περίπτωση του δικού μας συστήματος, είναι η ενσωματωμένη εφαρμογή που εκτελείται στο DA14580.

- **Weather Collector Role**: Είναι ένας GAP central GATT client. Αποτελεί το τμήμα του συστήματος, που δέχεται μετρήσεις καιρικών δεδομένων από τον Weather Station και τις παρουσιάζει στον χρήστη. Επίσης, δίνει τη δυνατότητα στον χρήστη να ρυθμίσει τις παραμέτρους λειτουργίας του συστήματος και να εκτελέσει λειτουργίες που προβλέπονται από το προφίλ. Στην περίπτωση του δικού μας συστήματος, είναι το Android app που εκτελείται σε smartphone ή tablet.

6.2.1 Υπηρεσίες αισθητήρων

Έχουμε, λοιπόν, από μία υπηρεσία για κάθε αισθητήρα που περιέχεται στο σύστημα (sensor service). Για παράδειγμα, ο Weather Station μπορεί να περιέχει τις εξής υπηρεσίες:

- Temperature Sensor Service
- Humidity Sensor Service
- Barometric Pressure Sensor Service
- Wind Speed Sensor Service
- Wind Direction Sensor Service
- Ambient Light Sensor Service
- UV Index Sensor Service

Οι υπηρεσίες αυτές παρουσιάζουν παρόμοια συμπεριφορά και ορίζονται, όπως θα δούμε παρακάτω, με παρόμοιο τρόπο. Κάθε υπηρεσία δίνει στους χρήστες τις εξής δυνατότητες:

- Ανάκτηση της τελευταίας μέτρησης που έγινε από τον αισθητήρα.
- Ρύθμιση του αισθητήρα, ώστε να πραγματοποιεί περιοδικές μετρήσεις ανά συγκεκριμένο χρονικό διάστημα, στέλνοντας τα αποτελέσματα.
- Ενεργοποίηση ή απενεργοποίηση της παραπάνω αποστολής, αν, για παράδειγμα, ο collector δε θέλει να λαμβάνει δεδομένα για κάποιο χρονικό διάστημα.
- Εκτέλεση εντολών στον αισθητήρα.

Για τις περιοδικές μετρήσεις, ο χρήστης μπορεί να ορίσει το χρονικό διάστημα, που μεσολαβεί ανάμεσα στις μετρήσεις. Οι δυνατές τιμές είναι, με ανάλυση ενός δευτερολέπτου, από ένα, οπότε έχουμε μία μέτρηση ανά δευτερόλεπτο έως 86400 δευτερόλεπτα, οπότε έχουμε μία μέτρηση ανά ημέρα. Η τιμή μηδέν είναι ειδική τιμή, η οποία σημαίνει απενεργοποίηση των περιοδικών μετρήσεων.

Οι εντολές που μπορούν να εκτελεστούν από τον αισθητήρα είναι οι ακόλουθες:

- Άμεση εκκίνηση μέτρησης (“Update Now”) και αποστολή του αποτελέσματος (αν οι αποστολές είναι ενεργοποιημένες). Η διαδικασία είναι ανεξάρτητη των περιοδικών μετρήσεων που μπορεί να εκτελεί ο αισθητήρας.
- Ενεργοποίηση του αισθητήρα.
- Απενεργοποίηση του αισθητήρα.
- Επαναφορά του αισθητήρα (Reset).

Η ικανότητα ενεργοποίησης και απενεργοποίησης αισθητήρων δίνει στον χρήστη τη δυνατότητα να ορίζει ποιους αισθητήρες θέλει κάθε στιγμή να έχει ενεργούς. Με αυτό τον τρόπο, αισθητήρες των οποίων οι μετρήσεις δεν ενδιαφέρουν τον χρήστη, μπορούν να απενεργοποιηθούν, για την εξοικονόμηση ενέργειας στη συσκευή καταγραφής. Για να λειτουργήσει όμως αυτό, πρέπει ο συγκεκριμένος αισθητήρας να μπορεί να απενεργοποιηθεί ανεξάρτητα από το υπόλοιπο σύστημα, δηλαδή να έχει δικό του power domain. Διαφορετικά, ο χρήστης θα μπορούσε απλώς να σταματήσει τις περιοδικές μετρήσεις του αισθητήρα. Η ικανότητα απενεργοποίησης αισθητήρων δεν είναι υποχρεωτικό να υποστηρίζεται από τη συσκευή καταγραφής. Σε κάθε περίπτωση, όμως, αν η συγκεκριμένη εντολή αναγνωρίζεται από τη συσκευή, ένας απενεργοποιημένος αισθητήρας δεν μπορεί να εκτελεί μετρήσεις, ούτε περιοδικές, ούτε μετά από εντολή του χρήστη.

Παρόμοια λογική ισχύει και για την επαναφορά. Η συγκεκριμένη λειτουργία υπάρχει για την περίπτωση που ο αισθητήρας παρουσιάζει δυσλειτουργία ή μη αναμενόμενη συμπεριφορά. Επιτρέποντας την επαναφορά του συγκεκριμένου μόνο αισθητήρα, το σύστημα μπορεί να αποφύγει την ανάγκη συνολικής επαναφοράς με επανεκκίνησή του. Όπως και πριν, δεν είναι απαραίτητο αυτή η λειτουργία να υποστηρίζεται από τη συσκευή καταγραφής. Ο τρόπος επαναφοράς είναι επίσης εξαρτώμενος από την υλοποίηση. Μπορεί δηλαδή να υποστηρίζεται από τον συγκεκριμένο αισθητήρα δυνατότητα software reset ή, στην περίπτωση που αυτός έχει δικό του power domain, μπορεί να γίνεται επανεκκίνησή του, κλείνοντας και ανοίγοντας την παροχή ενέργειας.

Η συσκευή καταγραφής, λοιπόν, λειτουργεί ως GATT server, ο οποίος περιέχει από μία υπηρεσία για κάθε αισθητήρα που περιλαμβάνεται σε αυτή. Ο αριθμός και το είδος των αισθητήρων ορίζεται από τη συγκεκριμένη υλοποίηση και δεν είναι υποχρεωτικός. Όμως, είναι καλό να υπάρχουν τουλάχιστον οι αισθητήρες για τα βασικά μεγέθη, δηλαδή θερμοκρασία, υγρασία, ατμοσφαιρική πίεση.

6.2.2 Γενική υπηρεσία σταθμού

Επιπλέον των υπηρεσιών για κάθε αισθητήρα χωριστά, αποφασίσαμε να ορίσουμε μία ακόμα υπηρεσία (αναφέρεται ως weather station service ή master service), η οποία κάνει include όλες τις υπηρεσίες αισθητήρων που περιλαμβάνονται στη συσκευή, ορίζοντας, πάνω από αυτές, νέες διαδικασίες και λειτουργικότητα. Η υπηρεσία αυτή έχει παρόμοια δομή με τις υπηρεσίες των αισθητήρων.

Η κύρια χρήση, για την οποία προορίζεται, είναι η αποστολή δεδομένων πολλών μετρήσεων ταυτόχρονα, στο ίδιο πακέτο BLE. Αυτό έχει ως αποτέλεσμα την αποφυγή αποστολής διαφορετικού πακέτου για κάθε μέτρηση, αν τα δεδομένα πολλών μετρήσεων είναι διαθέσιμα την ίδια στιγμή. Έτσι, μειώνεται το κόστος επικοινωνίας μεταξύ των συσκευών. Αυτό είναι σημαντικό, γιατί για κάθε αποστολή μιας μέτρησης, με μέγεθος 2 ως 3 bytes, το overhead της αποστολής είναι (χωρίς κρυπτογράφηση):

10 bytes (Link Layer) + 4 bytes (L2CAP) + 3 bytes (ATT Notification) = 17 bytes

Αντί, λοιπόν, να σταλούν διαφορετικά πακέτα με 2 ως 3 bytes δεδομένων στο καθένα, μπορεί να σταλεί μόνο ένα με όλα αυτά τα δεδομένα, συν ένα ως δύο bytes για τον καθορισμό των μετρήσεων που περιέχονται σε αυτό. Το όφελος γίνεται ακόμη μεγαλύτερο, αν συνυπολογίσουμε και το γεγονός ότι, για κάθε αποστολή πακέτου, η συσκευή θα πρέπει να λάβει και την επιβεβαίωση από το Link Layer, ελέγχοντας το μέσο για αποστολή πακέτων από την απομακρυσμένη συσκευή. Όπως ισχύει για κάθε αισθητήρα, έτσι και στη γενική υπηρεσία, είναι δυνατόν να απενεργοποιηθεί η αποστολή των παραπάνω δεδομένων, με κατάλληλη ρύθμιση στη συσκευή καταγραφής.

Παράλληλα, η γενική υπηρεσία χρησιμοποιεί ένα κεντρικό ρολόι, μέσω του οποίου μπορούν να γίνονται περιοδικές μετρήσεις ταυτόχρονα, σε περισσότερους από έναν αισθητήρες. Οι δυνατές τιμές της περιόδου είναι όμοιες με αυτές που ισχύουν για τους αισθητήρες. Δίνεται με αυτόν τρόπο στον χρήστη η δυνατότητα να ελέγξει κεντρικά όλους τους αισθητήρες, χωρίς, όπως θα δούμε παρακάτω, να του αφαιρείται η δυνατότητα ρύθμισης κάθε αισθητήρα χωριστά.

Η γενική υπηρεσία επεκτείνει τις εντολές που ορίζονται για κάθε αισθητήρα, ενώ προσθέτει και νέες, που παρέχουν δυνατότητα ρύθμισης της επιπλέον λειτουργικότητας. Οι εντολές που ορίζονται είναι οι εξής:

- Άμεση εκκίνηση μέτρησης σε συγκεκριμένο αισθητήρα ή σε όλους τους αισθητήρες. Αποστολή του αποτελέσματος, αν οι αποστολές είναι ενεργοποιημένες.
- Ενεργοποίηση συγκεκριμένου αισθητήρα ή όλων των αισθητήρων.
- Απενεργοποίηση συγκεκριμένου αισθητήρα ή όλων των αισθητήρων.
- Επαναφορά συγκεκριμένου αισθητήρα ή όλων των αισθητήρων (Reset).
- Ενεργοποίηση του ελέγχου, από τη γενική υπηρεσία, συγκεκριμένου αισθητήρα ή όλων των αισθητήρων.
- Απενεργοποίηση του ελέγχου, από τη γενική υπηρεσία, συγκεκριμένου αισθητήρα ή όλων των αισθητήρων.
- Ενεργοποίηση της δυνατότητας αποστολής των μετρήσεων συγκεκριμένου αισθητήρα ή όλων των αισθητήρων, μέσω της γενικής υπηρεσίας.
- Απενεργοποίηση της δυνατότητας αποστολής των μετρήσεων συγκεκριμένου αισθητήρα ή όλων των αισθητήρων, μέσω της γενικής υπηρεσίας.
- Ανάκτηση των ρυθμίσεων των παραπάνω για ένα συγκεκριμένο αισθητήρα.
- Συγχρονισμός περιοδικών μετρήσεων των αισθητήρων.

Στις παραπάνω εντολές, εκτός αυτής του συγχρονισμού, παρέχεται από τον collector ένα αναγνωριστικό, το οποίο και ορίζει τον αισθητήρα για τον οποίο προορίζεται η λειτουργία. Στην περίπτωση που δεν υπάρχει αυτό το αναγνωριστικό, θεωρείται ότι η λειτουργία αφορά όλους τους αισθητήρες.

Οι πρώτες εντολές λειτουργούν όπως στις υπηρεσίες των αισθητήρων. Οι νέες εντολές δίνουν τη δυνατότητα στον χρήστη να ρυθμίσει τις επιπλέον λειτουργίες που παρέχονται από τη γενική υπηρεσία. Έτσι, μπορεί να εξαιρέσει συγκεκριμένους αισθητήρες από τις περιοδικές μετρήσεις της γενικής υπηρεσίας. Επίσης, μπορεί να ορίσει ποιες μετρήσεις αισθητήρων μπορούν να στέλνονται μέσω της γενικής υπηρεσίας. Οι τρέχουσες τιμές των δύο παραπάνω ρυθμίσεων μπορούν να ανακτηθούν με την αντίστοιχη εντολή. Τέλος, ο χρήστης μπορεί να ζητήσει από τη συσκευή καταγραφής τον συγχρονισμό των περιοδικών μετρήσεων. Αυτό χρησιμεύει, για παράδειγμα, στην περίπτωση, που εξαιτίας διαφορετικών χρονικών σημείων ρύθμισης, οι αισθητήρες είναι αποσυγχρονισμένοι. Ο συγχρονισμός μπορεί να μειώσει το κόστος επικοινωνίας, αφού, μετά τον συγχρονισμό, οι μετρήσεις των αισθητήρων μπορούν να ξεκινούν ταυτόχρονα, οπότε και να στέλνονται ταυτόχρονα (συχνότερα ή πάντα, ανάλογα με τις περιόδους μέτρησης).

Το πρότυπο του BLE ορίζει ότι μια υπηρεσία, που κάνει include άλλες υπηρεσίες, δεν επιτρέπεται να αλλάξει τη συμπεριφορά τους. Αυτό ελήφθη υπόψη κατά το σχεδιασμό της γενικής υπηρεσίας. Για τη συμπεριφορά του συστήματος ισχύουν οι παρακάτω κανόνες:

- Μόνο μετρήσεις αισθητήρων, που βρίσκονται υπό τον έλεγχο του master service, μπορούν να ξεκινήσουν από αυτό, κατά τις περιοδικές μετρήσεις του.
- Αν ένας αισθητήρας έχει ορισμένη (μη μηδενική) περίοδο μέτρησης, τότε αυτή έχει προτεραιότητα. Δηλαδή, ακόμα και αν ο αισθητήρας είναι υπό τον έλεγχο του master service, δε συμμετέχει στις περιοδικές μετρήσεις του τελευταίου, αλλά ακολουθεί τη δική του περίοδο λειτουργίας.
- Μόνο μετρήσεις αισθητήρων, με ενεργοποιημένη τη δυνατότητα αποστολής τους μέσω του master service, μπορούν να περιέχονται στις μετρήσεις που στέλνονται από αυτό (αν αυτές είναι ενεργοποιημένες).
- Αν ένας αισθητήρας έχει ενεργοποιημένη τη ρύθμιση αποστολής μετρήσεων, τότε οι μετρήσεις αυτού του αισθητήρα στέλνονται πάντα σε ξεχωριστό πακέτο, και όχι μέσω του master service.

Βλέπουμε, δηλαδή, ότι οι ρυθμίσεις κάθε αισθητήρα έχουν μεγαλύτερη προτεραιότητα σε σχέση με τις ρυθμίσεις της γενικής υπηρεσίας. Αυτό είναι απαραίτητο, ώστε οι υπηρεσίες των αισθητήρων να λειτουργούν με την ίδια συμπεριφορά, ανεξάρτητα από τη χρήση τους από τη γενική υπηρεσία.

Αν και δε σχεδιάστηκε για αυτόν τον λόγο, ένα επιπλέον πλεονέκτημα της χρήσης της γενικής υπηρεσίας είναι η δυνατότητα αντιμετώπισης περιορισμών του BLE stack στο Android. Όπως είδαμε στο προηγούμενο κεφάλαιο, το Android BLE stack (API level 19) έχει ένα όριο στον αριθμό των notifications ή indications, τα οποία μπορούν να είναι ενεργοποιημένα, τοπικά, ταυτόχρονα. Για να μπορεί κάθε αισθητήρας να στέλνει τις μετρήσεις του ξεχωριστά, και αυτές να λαμβάνονται από την εφαρμογή, πρέπει το αντίστοιχο notification να είναι ενεργοποιημένο. Αν έχουμε πολλούς αισθητήρες, αυτοί δε θα μπορούν να στέλνουν μετρήσεις ταυτόχρονα. Μέσω του master service, το οποίο χρειάζεται ένα μόνο notification ενεργοποιημένο για να στείλει τις μετρήσεις, μπορούμε να παρακάμψουμε τον παραπάνω περιορισμό.

Για τον ορισμό των χαρακτηριστικών, των υπηρεσιών και, τελικά, του ίδιου του προφίλ, χρησιμοποιήσαμε τη σύνταξη XML, που ορίζεται από το SIG για τον σκοπό αυτό.

6.3 Χαρακτηριστικά

Σε αυτή την υπο-ενότητα, περιγράφουμε τα χαρακτηριστικά των υπηρεσιών, τη μορφή τους και τις δυνατές τιμές τους. Οι υπηρεσίες των αισθητήρων και η γενική υπηρεσία έχουν παρόμοια δομή. Κάθε μία από αυτές περιέχει τρία χαρακτηριστικά, τα οποία είναι τα εξής:

- Measurement
- Measurement Interval
- Control Point

6.3.1 Measurement

Στην περίπτωση των sensor services, το χαρακτηριστικό measurement περιέχει το αποτέλεσμα της τελευταίας μέτρησης που έγινε από τον αισθητήρα, το οποίο μπορεί να διαβαστεί (GATT Read) ή να σταλεί (μέσω Gatt Notification). Στην περίπτωση του master service, το measurement δεν μπορεί να διαβαστεί, αλλά μόνο να σταλεί. Όπως ορίζεται στο πρότυπο του BLE, εφόσον χρησιμοποιούνται notifications, δίνεται στον collector η δυνατότητα να ελέγξει την αποστολή των notifications, μέσω ενός Client Characteristic Configuration Descriptor. Η συγκεκριμένη ρύθμιση είναι αυτή που ενεργοποιεί την αποστολή μετρήσεων από κάθε υπηρεσία.

Για κάθε αισθητήρα ορίσαμε ένα ξεχωριστό measurement characteristic. Η μορφή της τιμής του measurement characteristic εξαρτάται από την υπηρεσία στην οποία ανήκει. Οι μονάδες μέτρησης και τα μεγέθη τιμών αναφέρονται στον ακόλουθο πίνακα:

Characteristic	**Format**	**Unit**
Temperature Measurement	SFLOAT (2 bytes)	Celsius / Fahrenheit
Humidity Measurement	SFLOAT (2 bytes)	Percentage (%) : 0 – 100
Barometric Pressure Measurement	uint24 (3 bytes)	Pascal (Pa)
Ambient Light Measurement	uint24 (3 bytes)	lux
UV Index Measurement	uint16 (2 bytes)	Unitless actual_value = value * 10^{-2}
Wind Speed Measurement	uint16 (2 bytes)	km/h actual_value = value * 10^{-2}
Wind Direction Measurement	uint8 (1 byte)	Unitless (enumeration)

Η μονάδα SFLOAT (short float) ορίζεται στο πρότυπο IEEE-11073 και είναι ένας αριθμός κινητής υποδιαστολής των 16 bit με τη μορφή:

Exponent 4 bit singed	**Mantissa** 12 bit singed

Η τιμή ενός SFLOAT προκύπτει ως εξής:

$$value = mantissa \times 10^{\exp onent}$$

Το χαρακτηριστικό measurement για sensor services μπορεί, προαιρετικά, να περιέχει ένα Characteristic Presentation Format Descriptor. Σε αυτό περιλαμβάνονται οι παραπάνω πληροφορίες σχετικά με την τιμή του, ώστε, ακόμα και μη συμβατοί clients, να μπορούν να αναγνωρίσουν ορισμένα από τα μεγέθη που παρέχονται. Εφόσον τα μεγέθη είναι καθορισμένα, ένας συμβατός client (collector) γνωρίζει εξαρχής τον τύπο των τιμών. Η μόνη περίπτωση, στην οποία είναι απαραίτητο να υπάρχει το Presentation Format, είναι όταν η μέτρηση της θερμοκρασίας είναι σε Fahrenheit. Αυτό πρέπει να δηλωθεί μέσω του Presentation Format. Αν δεν υπάρχει Presentation Format, τότε η μέτρηση θεωρείται ότι είναι σε Celsius. Για παράδειγμα, μπορούμε να έχουμε:

Characteristic	Format	Exponent	Unit	Namespace	Description
Temperature Measurement in Celcius	0x16 SFLOAT	0	0x272F Celcius	SIG (1)	unknown (0)
Temperature Measurement in Fahrenheit	0x16 SFLOAT	0	0x27AC Fahrenheit	SIG (1)	unknown (0)
Humidity Measurement	0x16 SFLOAT	0	0x27AD percentage	SIG (1)	unknown (0)
UV Index Measurement	0x06 uint16	0xFE (-2)	0x2700 unitless	SIG (1)	unknown (0)

Ακολουθούν τμήματα των XML αρχείων, που ορίζουν τα χαρακτηριστικά temperature και humidity measurement.

- **Temperature Sensor Measurement**

```
<Characteristic
...
    type="gr.ntua.microlab.characteristic.temperature_sensor_measurement"
    uuid="DC981101-F292-11E3-B75F-002215F5EF22"
    name="Temperature Sensor Measurement">
...
<Value>
<Field name="Measurement (Celsius)">
<InformativeText>
       If 'Measurement (Fahrenheit)' field is not present,
       this field is mandatory, else it is excluded.
</InformativeText>
<Requirement>Conditional</Requirement>
<Format>SFLOAT</Format>
<Unit>org.bluetooth.unit.thermodynamic_temperature.degree_celsius</Unit>
</Field>

<Field name="Measurement (Fahrenheit)">
<InformativeText>
       If 'Measurement (Celsius)' field is not present,
       this field is mandatory, else it is excluded.
</InformativeText>
<Requirement>Conditional</Requirement>
<Format>SFLOAT</Format>
<Unit>org.bluetooth.unit.thermodynamic_temperature.degree_fahrenheit</Unit>
</Field>
</Value>
...
</Characteristic>
```

- **Humidity Sensor Measurement**

```
<Characteristic
...
    type="gr.ntua.microlab.characteristic.humidity_sensor_measurement"
    uuid="DC981201-F292-11E3-B75F-002215F5EF22"
    name="Humidity Sensor Measurement">
...
    <Value>
```

```
        <Field name="Measurement">
            <Requirement>Mandatory</Requirement>
            <Format>SFLOAT</Format>
            <Unit>org.bluetooth.unit.percentage</Unit>
        </Field>
    </Value>
...
</Characteristic>
```

Το measurement του master service δεν έχει Presentation Format. Έχει μεταβλητό μέγεθος μέχρι (MTU - 3) bytes, ώστε να μπορεί να σταλεί σε ένα πακέτο. Το μέγεθός του εξαρτάται από τον αριθμό και το είδος των μετρήσεων που περιέχονται κάθε φορά σε αυτό. Για τον καθορισμό των παραπάνω, η τιμή του ξεκινά με ένα ή δύο bytes με σημαίες, καθεμιά από τις οποίες δηλώνει την παρουσία, στα επόμενα bytes, της αντίστοιχης μέτρησης, όπως ορίζεται στο προφίλ. Ακολουθεί μέρος του XML αρχείου, που ορίζει το χαρακτηριστικό:

- **Weather Station Measurement**

```
<Characteristic
...
    type="gr.ntua.microlab.characteristic.weather_station_measurement"
    uuid="DC981001-F292-11E3-B75F-002215F5EF22"
    name="Weather Station Measurement">
...
<Field name="Flags">
<InformativeText>
    Specifies which data are present in this measurement.
</InformativeText>
<Requirement>Mandatory</Requirement>
<Format>8bit</Format>
<BitField>
    <Bit index="0" size="1" name="Temperature">
        <Enumerations>
        <Enumeration key="0"
                     value="Temperature field not present" />
        <Enumeration key="1"
                     value="Temperature field present"
                     requires="C1" />
        </Enumerations>
    </Bit>
    <Bit index="1" size="1" name="Humidity">
        <Enumerations>
        <Enumeration key="0"
                     value="Humidity field not present" />
        <Enumeration key="1"
                     value="Humidity field present"
                     requires="C2" />
        </Enumerations>
    </Bit>
    <Bit index="2" size="1" name="Barometric Pressure">
        <Enumerations>
        <Enumeration key="0"
                     value="Barometric Pressure field not present" />
        <Enumeration key="1"
                     value="Barometric Pressure field present"
                     requires="C3" />
        </Enumerations>
    </Bit>
    ...
</BitField>
</Field>

...

<Field name="Temperature">
```

```
<Requirement>C1</Requirement>
<Reference>
gr.ntua.microlab.characteristic.temperature_sensor_measurement
</Reference>
</Field>

<Field name="Humidity">
<Requirement>C2</Requirement>
<Reference>
gr.ntua.microlab.characteristic.humidity_sensor_measurement
</Reference>
</Field>

<Field name="Barometric Pressure">
<Requirement>C3</Requirement>
<Reference>
gr.ntua.microlab.characteristic.barometric_pressure_sensor_measurement
</Reference>
</Field>

...
</Characteristic>
```

6.3.2 Measurement Interval

Το χαρακτηριστικό measurement interval ενεργοποιεί τις περιοδικές μετρήσεις και καθορίζει την περίοδό τους. Έχει την ίδια μορφή σε όλες τις υπηρεσίες. Παρότι στα ορισμένα από το SIG χαρακτηριστικά, περιέχεται ένα χαρακτηριστικό αυτού του τύπου (org.bluetooth.characteristic.measurement_interval, με UUID 2A21), αποφασίσαμε να ορίσουμε καινούρια χαρακτηριστικά, τόσο για την υποστήριξη μεγαλύτερων τιμών, όσο και για τον διαχωρισμό των measurement intervals των διάφορων αισθητήρων. Αν χρησιμοποιούσαμε το SIG measurement interval, θα είχε το ίδιο UUID σε όλους τους αισθητήρες. Αυτό δεν απαγορεύεται από το πρότυπο, αλλά μπορεί να περιπλέξει τον διαχωρισμό τους από τον collector.

Ορίσαμε, λοιπόν, από ένα measurement interval χαρακτηριστικό για κάθε υπηρεσία. Το measurement interval μπορεί να διαβαστεί (GATT Read) ή να γραφτεί (GATT Write). Οι δυνατές τιμές, όπως έχουμε ήδη αναφέρει, είναι από 1 έως 86400, σε δευτερόλεπτα. Η τιμή μηδέν απενεργοποιεί τις περιοδικές μετρήσεις. Το format της τιμής είναι uint24 (3 bytes). Μέσω αυτού του χαρακτηριστικού, μπορεί ο χρήστης να ρυθμίσει τις περιοδικές μετρήσεις, τόσο του master service, όσο και κάθε αισθητήρα ξεχωριστά. Για τη συμπεριφορά του συστήματος σε σχέση με την τιμή του measurement interval, ισχύουν οι κανόνες που αναφέρθηκαν παραπάνω.

Εφόσον το measurement interval έχει την ίδια μορφή σε όλα τα services, το ορίζουμε για το master service και όλα τα υπόλοιπα measurement intervals χρησιμοποιούν μια αναφορά σε αυτό.

- **Weather Station Measurement Interval**

```
<Characteristic
...
type="gr.ntua.microlab.characteristic.weather_station_measurement_interval"
uuid="DC981002-F292-11E3-B75F-002215F5EF22"
name="Weather Station Measurement Interval">
...
    <InformativeText>
        <Abstract>
```

```
            The Weather Station Measurement Interval characteristic defines
            the time between measurements.
        </Abstract>
        <Summary>
            This characteristic is capable of representing values from 1
            second to 86400 seconds which is equal to 1 day.
        </Summary>
    </InformativeText>
    <Value>
        <Field name="Measurement Interval">
            <Requirement>Mandatory</Requirement>
            <Format>uint24</Format>
            <Unit>org.bluetooth.unit.time.second</Unit>
            <Minimum>1</Minimum>
            <Maximum>86400</Maximum>
            <AdditionalValues>
                <Enumeration key="0" value="No periodic measurement" />
            </AdditionalValues>
        </Field>
    </Value>
...
</Characteristic>
```

- **Temperature Sensor Measurement Interval**

```
<Characteristic
   type=
   "gr.ntua.microlab.characteristic.temperature_sensor_measurement_interval"
   uuid="DC981102-F292-11E3-B75F-002215F5EF22"
   name="Temperature Sensor Measurement Interval">
...
    <Value>
        <Field name="Measurement Interval">
            <Requirement>Mandatory</Requirement>
            <Reference>
        gr.ntua.microlab.characteristic.weather_station_measurement_interval
            </Reference>
        </Field>
    </Value>
...
</Characteristic>
```

6.3.3 Control Point

Το control point χαρακτηριστικό υπάρχει σε όλες τις υπηρεσίες και χρησιμοποιείται για την αποστολή εντολών στη συσκευή καταγραφής καιρού, με σκοπό την εκτέλεση λειτουργιών από αυτή. Οι εντολές είναι αυτές που αναφέρθηκαν παραπάνω. Για κάθε εντολή ορίζεται στο προφίλ μία τιμή, που πρέπει να γραφτεί από τον collector στην τιμή του control point χαρακτηριστικού, προκειμένου να ξεκινήσει η λειτουργία που προβλέπεται για αυτή. Μετά την ολοκλήρωση της λειτουργίας, η συσκευή καταγραφής καιρού στέλνει απάντηση στον collector με τα αποτελέσματα. Η απάντηση στέλνεται μέσω GATT Indication του control point χαρακτηριστικού. Το προφίλ ορίζει τις τιμές επιστροφής.

Ορίσαμε, λοιπόν, από ένα control point χαρακτηριστικό για κάθε αισθητήρα. Οι λειτουργίες του GATT, που υποστηρίζονται στο control point χαρακτηριστικό, είναι οι GATT Write, GATT Write Without Response και GATT Indication. Το Write Without Response μπορεί να χρησιμοποιηθεί μόνο για την εντολή “Update Now”. Δεν επιτρέπεται η χρήση του σε άλλες εντολές. Το control point χαρακτηριστικό περιλαμβάνει ένα Client Characteristic Configuration Descriptor για την ενεργοποίηση των indications. Προκειμένου

να λάβει τα αποτελέσματα, ο collector πρέπει πρώτα να ενεργοποιήσει τα indications. Διαφορετικά, η συσκευή απαντά με μήνυμα λάθους. Επίσης, δεν επιτρέπεται η αποστολή νέας εντολής, πριν ολοκληρωθεί η προηγούμενη και ληφθεί το αντίστοιχο indication. Αν ο collector προσπαθήσει να στείλει νέα εντολή πριν λάβει την απάντηση, τότε η συσκευή απαντά με μήνυμα λάθους. Έτσι, δημιουργείται ένας μηχανισμός flow control μεταξύ weather station και collector, και ορίζονται transactions με παρόμοιο τρόπο όπως στο ATT. Αυτή η τεχνική χρησιμοποιείται σε πολλά από τα control point characteristics των adopted profiles του SIG. Αξίζει, εδώ, να σημειωθεί ότι το ATT, και κατ' επέκταση το GATT, είναι, σε γενικές γραμμές, stateless πρωτόκολλα. Πέρα από τα transactions και την ουρά του Prepare Write, δεν διατηρείται κάποιο state. Αυτό απλοποιεί τις διαδικασίες και τις απαιτήσεις μνήμης. Παρατηρούμε, λοιπόν, πώς μπορούμε, χρησιμοποιώντας κατάλληλες GATT διαδικασίες, να ορίσουμε state πάνω στη stateless λειτουργία των ATT/GATT.

Στο σύστημα καταγραφής καιρού, υπάρχουν δύο δυνατότητες σε σχέση με το control point χαρακτηριστικό:

- Ο collector στέλνει μία εντολή "Update Now", γράφοντας κατάλληλη τιμή στο control point, με τη λειτουργία Write Without Response του GATT. Ο weather station ξεκινά τη μέτρηση στους αισθητήρες, που ορίζονται από την εντολή (έναν ή περισσότερους), και, στη συνέχεια, στέλνει τα αποτελέσματα των μετρήσεων μέσω notification του χαρακτηριστικού measurement. Δε στέλνεται indication.

- Ο collector στέλνει μία εντολή, εκτός της "Update Now", γράφοντας κατάλληλη τιμή στο control point, με τη λειτουργία Write του GATT. Ο weather station ελέγχει το είδος της εντολής και εκκινεί την αντίστοιχη λειτουργία. Μετά την ολοκλήρωσή της, ο weather station στέλνει το αποτέλεσμα (π.χ. επιτυχία ή αποτυχία), μέσω GATT Indication του control point χαρακτηριστικού. Αν υπάρχει κάποιο λάθος στην εντολή, ή αυτή δεν υποστηρίζεται, ο weather station στέλνει κατάλληλο μήνυμα λάθους με τον ίδιο τρόπο.

Η μορφή της τιμής του control point χαρακτηριστικού είναι διαφορετική στην αποστολή των εντολών και των αποτελεσμάτων. Στις εντολές, ξεκινά με ένα byte opcode, το οποίο καθορίζει την εντολή προς εκτέλεση. Αυτό, μπορεί προαιρετικά, και μόνο στην περίπτωση του master service control point, να ακολουθείται από το αναγνωριστικό του αισθητήρα για τον οποίο προορίζεται η εντολή. Οι τιμές αυτές ορίζονται στο προφίλ. Στα αποτελέσματα, ξεκινά με ένα byte που καθορίζει το είδος των αποτελεσμάτων. Υπάρχουν δύο είδη αποτελεσμάτων. Στις περισσότερες εντολές τα αποτελέσματα δηλώνουν απλώς την επιτυχία ή αποτυχία της λειτουργίας, ή πιθανά σφάλματα στην εντολή. Σε αυτές τις περιπτώσεις, το πρώτο byte έχει την τιμή "Response Code", ενώ ένα δεύτερο byte καθορίζει το αποτέλεσμα της λειτουργίας. Στην περίπτωση, όμως, της εντολής ανάκτησης των ρυθμίσεων ενός αισθητήρα από το master service, αν δεν προκύψει κάποιο σφάλμα, χρησιμοποιείται άλλη τιμή στο πρώτο byte ("Get Sensor Master Configuration Response"), και το δεύτερο περιέχει τις ζητούμενες ρυθμίσεις. Οι τιμές αυτές ορίζονται επίσης στο προφίλ.

Εφόσον το control point έχει την ίδια μορφή σε όλα τα sensor services, ορίζουμε ένα Generic Sensor Control Point, και όλα τα υπόλοιπα control points χρησιμοποιούν μια αναφορά σε αυτό.

- **Generic Sensor Control Point**

```
<Characteristic
...
type="gr.ntua.microlab.characteristic.generic_sensor_control_point"
uuid="DC980FFF-F292-11E3-B75F-002215F5EF22"
```

```
name="Generic Sensor Control Point">
...
    <Value>
        <Field name="Op Code">
            <InformativeText>
                The Op Code specifies the sensor operation.
            </InformativeText>
            <Requirement>Mandatory</Requirement>
            <Format>uint8</Format>
            <Enumerations>
                <Enumeration key="0"
                    value="Nop"
                    description="No operation" />
                <Enumeration key="1"
                    value="Update Now"
                    description="Start a new measurement immediately" />
                <Enumeration key="2"
                    value="Start Sensor" />
                <Enumeration key="3"
                    value="Stop Sensor" />
                <Enumeration key="4"
                    value="Reset Sensor" />
                <Enumeration key="255"
                    value="Response Code"
                    requires="C1"/>
                <ReservedForFutureUse start="5" end="254" />
            </Enumerations>
        </Field>

        <Field name="Response Code">
            <InformativeText>
                Response Code Values. Present only when
                Op Code is "Response Code".
            </InformativeText>
            <Requirement>C1</Requirement>
            <Format>uint8</Format>
            <Enumerations>
                <Enumeration key="0"
                   value="Success"
                   description=
                   "Normal response for successful operation" />
                <Enumeration key="1"
                    value="Error"
                    description=
      "Normal response if unable to complete an operation for any reason" />
                <Enumeration key="2"
                    value="Op Code not supported"
                    description=
                    "Normal response if unsupported Op Code is received" />
                <ReservedForFutureUse start="3" end="255" />
            </Enumerations>
        </Field>
    </Value>
</Characteristic>
```

- **Humidity Sensor Control Point**

```
<Characteristic
...
    type="gr.ntua.microlab.characteristic.humidity_sensor_control_point"
    uuid="DC981203-F292-11E3-B75F-002215F5EF22"
    name="Humidity Sensor Control Point">
    <InformativeText>
        <Abstract>
            The Humidity Sensor Control Point characteristic is a control
            point characteristic that provides support for sensor operations
            and management.
```

```
        </Abstract>
        <Summary>
            This control point is used to initiate sensor operations
            or to access sensor management functions.
        </Summary>
    </InformativeText>
    <Value>
        <Field name="Control Point">
            <InformativeText>
                The Humidity Sensor Control Point is based
                on the Generic Sensor Control Point.
            </InformativeText>
            <Requirement>Mandatory</Requirement>
            <Reference>
                gr.ntua.microlab.characteristic.generic_sensor_control_point
            </Reference>
        </Field>
    </Value>
</Characteristic>
```

- **Weather Station Control Point**

Στην περίπτωση του master service, εξαιτίας των επεκτάσεων που προσφέρει, η μορφή και οι δυνατές τιμές είναι διαφορετικές. Αρχικά ορίζουμε τις επεκτάσεις. Πέρα από τις νέες λειτουργίες, προστίθεται στις προηγούμενες η δυνατότητα ορισμού του αισθητήρα για τον οποίο προορίζονται.

```
<Characteristic
...
    type="gr.ntua.microlab.characteristic.weather_station_control_point"
    uuid="DC981003-F292-11E3-B75F-002215F5EF22"
    name="Weather Station Control Point">
...
    <Value>
        <Field name="Op Code">
            <InformativeText>
                The Op Code specifies the weather station operation.
            </InformativeText>
            <Requirement>Mandatory</Requirement>
            <Format>uint8</Format>
            <Enumerations>
                <Enumeration key="0"
                    value="Nop"
                    description="No operation" />
                <Enumeration key="1"
                    value="Update Now"
                    description="Start a new measurement immediately"
                    requires="C2" />
                <Enumeration key="2"
                    value="Start Sensor"
                    requires="C2" />
                <Enumeration key="3"
                    value="Stop Sensor"
                    requires="C2" />
                <Enumeration key="4"
                    value="Reset Sensor"
                    requires="C2" />
                <Enumeration key="10"
                    value="Enable Sensor Measurement"
                    description="Enable master timer sensor measurement"
                    requires="C2" />
                <Enumeration key="11"
                    value="Disable Sensor Measurement"
                    description="Disable master timer sensor measurement"
                    requires="C2" />
```

```
                <Enumeration key="12"
                    value="Enable Sensor Notifications"
                    description="Enable master service sensor notifications"
                    requires="C2" />
                <Enumeration key="13"
                   value="Disable Sensor Notifications"
                   description="Disable master service sensor notifications"
                   requires="C2" />
                <Enumeration key="14"
                    value="Get Sensor Master Configuration"
                    requires="C2" />
                <Enumeration key="15"
                    value="Get Sensor Master Configuration Response"
                    requires="C3" />
                <Enumeration key="16"
                    value="Synchronize Sensors Measurements" />
                <Enumeration key="255"
                    value="Response Code"
                    requires="C1" />
                <ReservedForFutureUse start="5" end="9" />
                <ReservedForFutureUse start="17" end="254" />
            </Enumerations>
        </Field>
...
```

Ορίζεται επίσης ένα καινούριο Response Code, για την περίπτωση, που το αναγνωριστικό του αισθητήρα που δόθηκε, δεν αναγνωρίζεται.

```
...
<Field name="Response Code">
    <Enumerations>
    ...
    <Enumeration key="3"
        value="Invalid sensor ID"
        description="Normal response if invalid sensor ID is received" />
    <ReservedForFutureUse start="4" end="255" />
    </Enumerations>
</Field>
...
```

Επίσης ορίζονται τα αναγνωριστικά των αισθητήρων.

```
...
<Field name="Sensor ID">
...
    <InformativeText>
    Specifies the sensor for which the operation is intended.
    Present only when Op Code requires a sensor operand.
    If this field is required but not present then
    the value 255 ("All Sensors") is assumed.
    </InformativeText>
    <Requirement>C2</Requirement>
    <Format>uint8</Format>
    <Enumerations>
        <Enumeration key="0" value="Temperature Sensor" />
        <Enumeration key="1" value="Humidity Sensor" />
        <Enumeration key="2" value="Barometric Pressure Sensor" />
        ...
        <Enumeration key="255" value="All Sensors" />
        ...
     </Enumerations>
</Field>
...
```

Τέλος, ορίζεται το format της απάντησης στην εντολή “Get Sensor Master Configuration”. Αυτή είναι ένα bitfield, τα δύο LSB του οποίου δηλώνουν αν η αντίστοιχη ρύθμιση είναι ενεργοποιημένη.

```
...
<Field name="Sensor Master Configuration">
    <InformativeText>
    Contains the sensor's current master configuration.
    </InformativeText>
    <Requirement>C3</Requirement>
    <Format>uint8</Format>
    <BitField>
        <Bit index="0" size="1"
            name="Measurement Configuration">
            <Enumerations>
                <Enumeration key="0"
                    value="Master timer sensor measurement enabled" />
                <Enumeration key="1"
                    value="Master timer sensor measurement disabled" />
            </Enumerations>
        </Bit>
        <Bit index="1" size="1"
            name="Notifications Configuration">
            <Enumerations>
                <Enumeration key="0"
                    value="Master service sensor notifications enabled" />
                <Enumeration key="1"
                    value="Master service sensor notifications disabled" />
            </Enumerations>
        </Bit>
        <ReservedForFutureUse index="2" size="6" />
    </BitField>
</Field>
</Value>
...
```

6.4 Υπηρεσίες

Όπως αναφέραμε κατά την περιγραφή του GATT, μία υπηρεσία αποτελείται από ένα σύνολο χαρακτηριστικών, πάνω στα οποία ορίζει συγκεκριμένες συμπεριφορές, ενώ μπορεί να κάνει include άλλες υπηρεσίες. Στην περίπτωση του συστήματος καταγραφής καιρού, που υλοποιήσαμε, έχουμε, όπως δείξαμε παραπάνω, μία υπηρεσία για κάθε αισθητήρα που περιλαμβάνεται στο σύστημα, καθώς και τη γενική υπηρεσία (master service) του συστήματος, που κάνει include όλες τις άλλες, ορίζοντας επιπλέον λειτουργικότητα. Έχοντας ήδη περιγράψει τα χαρακτηριστικά από τα οποία αποτελούνται οι παραπάνω υπηρεσίες και την αναμενόμενη συμπεριφορά που δημιουργούν πάνω σε αυτά, αναφέρουμε, εδώ, συνοπτικά τα συστατικά στοιχεία κάθε υπηρεσίας και παραθέτουμε τμήματα των XML αρχείων με τον τυπικό ορισμό τους.

Οι υπηρεσίες που ορίσαμε, έχουν, όπως είδαμε, παρόμοια δομή και συμπεριφορά. Αυτό είναι αναμενόμενο, αφού οι υπηρεσίες ορίζονται με βάση τους αισθητήρες και αυτοί λειτουργούν με παρόμοιο τρόπο, ανεξάρτητα από το μέγεθος που μετράνε. Η μόνη ουσιαστική διαφορά μεταξύ των αισθητήρων είναι το format της τιμής μέτρησης. Τα συστατικά στοιχεία των υπηρεσιών παρουσιάζονται συνοπτικά στον ακόλουθο πίνακα.

Service Element Characteristic/Descriptor	Requirement / Comments	GATT sub-procedures
Measurement	Mandatory	Read (sensor only) Notifications
Client Characteristic Configuration Descriptor	Mandatory Ρύθμιση των GATT Notifications. Ενεργοποίηση / Απενεργοποίηση αποστολής αποτελεσμάτων μετρήσεων.	Read Write
Characteristic Presentation Format Descriptor	Optional Mandatory μόνο στην περίπτωση του temperature measurement, αν η μέτρηση είναι σε Fahrenheit.	Read
Measurement Interval	Mandatory	Read Write
Control Point	Mandatory	Write Write No Response Indications
Client Characteristic Configuration Descriptor	Mandatory Ρύθμιση των GATT Indications. Ενεργοποίηση / Απενεργοποίηση αποστολής αποτελεσμάτων λειτουργιών.	Read Write

Στο master service περιέχονται, επίσης, τα include definitions όλων των υπάρχοντων sensor services.

- **Humidity Sensor Service**

Ως παράδειγμα ορισμού ενός sensor service, χρησιμοποιούμε το Humidity Sensor Service. Αρχικά, ορίζεται ο τύπος και το UUID. Στον τυπικό ορισμό ενός service μέσω XML, μπορούν να δηλώνονται τυχόν Dependencies από άλλα services, ποια GATT sub-procedures πρέπει να υποστηρίζονται στις συσκευές (πέρα από όσα θεωρούνται υποχρεωτικά από το πρωτόκολλο), ο τύπος του Bluetooth (classic ή low energy) στον οποίο απευθύνεται, καθώς και επιπλέον κωδικοί σφάλματος που μπορούν να σταλούν στον client.

```
<Service
    ...
    type="gr.ntua.microlab.service.humidity_sensor"
    uuid="DC981200-F292-11E3-B75F-002215F5EF22"
    name="Humidity Sensor">
```

```
    <InformativeText>
        <Abstract>
            The Humidity Sensor service exposes the
            functionality of a humidity sensor.
        </Abstract>
        <Summary>
            This Humidity Sensor service provides access to humidity
            measurements and measurement settings, and provides support
            for sensor operations and management.
        </Summary>
    </InformativeText>

    <Dependencies>
        <Dependency>
            This service is not dependent upon any other services.
        </Dependency>
    </Dependencies>

    <GATTRequirements>
        <Requirement
            subProcedure="Write Characteristic Value">
            Mandatory</Requirement>
        <Requirement
            subProcedure="Notifications">
            Mandatory</Requirement>
        <Requirement
            subProcedure="Indications">
            Mandatory</Requirement>
        <Requirement
            subProcedure="Read Characteristic Descriptors">
            Mandatory</Requirement>
        <Requirement
            subProcedure="Write Characteristic Descriptors">
            Mandatory</Requirement>
    </GATTRequirements>

    <Transports>
        <Classic>false</Classic>
        <LowEnergy>true</LowEnergy>
    </Transports>

    <ErrorCodes>
        <ErrorCode name="Procedure Already in Progress" code="0xFE" />
        <ErrorCode
name="Client Characteristic Configuration Descriptor Improperly Configured"
code="0xFD" />
    </ErrorCodes>
...
```

Στη συνέχεια, ορίζονται τα χαρακτηριστικά και οι descriptors, με τις GATT λειτουργίες που πρέπει να υποστηρίζονται για καθένα από αυτά. Στην περίπτωση των χαρακτηριστικών, πρόκειται για τα characteristic properties, που περιέχονται στο characteristic declaration attribute.

```
...
<Characteristics>
    <Characteristic
        name="Humidity Measurement"
        type="gr.ntua.microlab.characteristic.humidity_sensor_measurement">
        <InformativeText>
            The result of the humidity sensor's last measurement.
        </InformativeText>

        <Requirement>Mandatory</Requirement>
```

```
        <Properties>
            <Read>Mandatory</Read>
            <Write>Excluded</Write>
            <WriteWithoutResponse>Excluded</WriteWithoutResponse>
            <SignedWrite>Excluded</SignedWrite>
            <ReliableWrite>Excluded</ReliableWrite>
            <Notify>Mandatory</Notify>
            <Indicate>Excluded</Indicate>
            <WritableAuxiliaries>Excluded</WritableAuxiliaries>
            <Broadcast>Excluded</Broadcast>
        </Properties>

        <Descriptors>
            <Descriptor
                name="Client Characteristic Configuration"
type="org.bluetooth.descriptor.gatt.client_characteristic_configuration">
                <Requirement>Mandatory</Requirement>
                <Properties>
                    <Read>Mandatory</Read>
                    <Write>Mandatory</Write>
                </Properties>
            </Descriptor>
            <Descriptor
                name="Characteristic Presentation Format"
type="org.bluetooth.descriptor.gatt.characteristic_presentation_format">
                <Requirement>Optional</Requirement>
                <Properties>
                    <Read>Mandatory</Read>
                    <Write>Excluded</Write>
                </Properties>
            </Descriptor>
        </Descriptors>
    </Characteristic>

    <Characteristic
        name="Measurement Interval"
type="gr.ntua.microlab.characteristic.humidity_sensor_measurement_interval">
        <InformativeText>
            The interval between consecutive humidity measurements.
            Setting interval to zero disables periodic measurements.
        </InformativeText>

        <Requirement>Mandatory</Requirement>
        <Properties>
            <Read>Mandatory</Read>
            <Write>Mandatory</Write>
            <WriteWithoutResponse>Excluded</WriteWithoutResponse>
            <SignedWrite>Excluded</SignedWrite>
            <ReliableWrite>Excluded</ReliableWrite>
            <Notify>Excluded</Notify>
            <Indicate>Excluded</Indicate>
            <WritableAuxiliaries>Excluded</WritableAuxiliaries>
            <Broadcast>Excluded</Broadcast>
        </Properties>
    </Characteristic>

    <Characteristic
        name="Humidity Sensor Control Point"
type="gr.ntua.microlab.characteristic.humidity_sensor_control_point">
        <InformativeText>
            This control point characteristic is used to
            initiate sensor operations (e.g. 'Update Now')
            or to access sensor management functions.
        </InformativeText>

        <Requirement>Mandatory</Requirement>
        <Properties>
            <Read>Excluded</Read>
```

```
            <Write>Mandatory</Write>
            <WriteWithoutResponse>Mandatory</WriteWithoutResponse>
            <SignedWrite>Excluded</SignedWrite>
            <ReliableWrite>Excluded</ReliableWrite>
            <Notify>Excluded</Notify>
            <Indicate>Mandatory</Indicate>
            <WritableAuxiliaries>Excluded</WritableAuxiliaries>
            <Broadcast>Excluded</Broadcast>
        </Properties>

        <Descriptors>
            <Descriptor
                name="Client Characteristic Configuration"
type="org.bluetooth.descriptor.gatt.client_characteristic_configuration">
                <Requirement>Mandatory</Requirement>
                <Properties>
                    <Read>Mandatory</Read>
                    <Write>Mandatory</Write>
                </Properties>
            </Descriptor>
        </Descriptors>
    </Characteristic>
</Characteristics>
...
```

- **Weather Station Service (master service)**

Ο ορισμός του master service είναι παρόμοιος με αυτόν των sensor services. Βασικότερη διαφορά είναι η ύπαρξη των included services, τα οποία δηλώνονται ως Dependencies και ως References. Τα References δηλώνονται ως Conditional, διότι θεωρούνται υποχρεωτικά μόνο στην περίπτωση που υπάρχει ο αντίστοιχος αισθητήρας, οπότε υπάρχει και το αντίστοιχο service.

```
<Service
    ...
    type="gr.ntua.microlab.service.weather_station"
    uuid="DC981000-F292-11E3-B75F-002215F5EF22"
    name="Weather Station">
...

    <Dependencies>
    ...
        <Dependency>
        Includes gr.ntua.microlab.service.temperature_sensor
        </Dependency>
        <Dependency>
        Includes gr.ntua.microlab.service.humidity_sensor
        </Dependency>
        <Dependency>
        Includes gr.ntua.microlab.service.barometric_pressure_sensor
        </Dependency>
    ...
    </Dependencies>
...
    <References>
    ...
        <Reference
            name="Temperature Sensor"
            type="gr.ntua.microlab.service.temperature_sensor">
            <Requirement> Conditional</Requirement>
        </Reference>
        <Reference
            name="Humidity Sensor"
            type="gr.ntua.microlab.service.humidity_sensor">
```

```
            <Requirement> Conditional</Requirement>
        </Reference>
        <Reference
            name="Barometric Pressure Sensor"
            type="gr.ntua.microlab.service.barometric_pressure_sensor">
            <Requirement> Conditional</Requirement>
        </Reference>
    ...
    </References>
...
```

Άλλη διαφορά είναι ότι το χαρακτηριστικό measurement δεν μπορεί να διαβαστεί, αλλά μόνο να σταλεί μέσω notification. Αυτή η συμπεριφορά επιλέχθηκε, γιατί το weather station measurement περιέχει απλώς τις τελευταίες μετρήσεις που στάλθηκαν μέσω αυτού, και όχι απαραίτητα τις τελευταίες μετρήσεις που έγιναν. Επίσης, ανάλογα με τις ρυθμίσεις, κάθε χρονική στιγμή μπορεί να περιέχει μετρήσεις από διαφορετικούς αισθητήρες. Σε κάθε περίπτωση, ο collector μπορεί να διαβάσει τις τιμές των τελευταίων μετρήσεων από τα measurements των αισθητήρων που τον ενδιαφέρουν.

```
...
<Characteristic
    name="Weather Station Measurement"
    type="gr.ntua.microlab.characteristic.weather_station_measurement">
    <InformativeText>
    The packed results of the weather station's sensors last measurements.
    </InformativeText>

    <Requirement>Mandatory</Requirement>
    <Properties>
        <Read>Excluded</Read>
        <Write>Excluded</Write>
        <WriteWithoutResponse>Excluded</WriteWithoutResponse>
        <SignedWrite>Excluded</SignedWrite>
        <ReliableWrite>Excluded</ReliableWrite>
        <Notify>Mandatory</Notify>
        <Indicate>Excluded</Indicate>
        <WritableAuxiliaries>Excluded</WritableAuxiliaries>
        <Broadcast>Excluded</Broadcast>
    </Properties>

    <Descriptors>
        <Descriptor
            name="Client Characteristic Configuration"
type="org.bluetooth.descriptor.gatt.client_characteristic_configuration">
            <Requirement>Mandatory</Requirement>
            <Properties>
                <Read>Mandatory</Read>
                <Write>Mandatory</Write>
            </Properties>
        </Descriptor>
    </Descriptors>
</Characteristic>
...
```

6.5 Προφίλ

Στη γενική περιγραφή του προφίλ είδαμε τους δύο ρόλους που ορίζονται σε αυτό, δηλαδή τους weather station role και weather collector role. Αναφέρουμε, εδώ, τις υπηρεσίες που πρέπει να υποστηρίζει ο κάθε ρόλος και παραθέτουμε τον τυπικό ορισμό του προφίλ μέσω XML.

Στο επίπεδο του GAP, ο weather station είναι peripheral, ενώ ο weather collector είναι central. Στο επίπεδο του GATT, ο weather station είναι server, ενώ ο weather collector είναι client. Ο weather station είναι, λοιπόν, ο ρόλος του προφίλ, όπου περιλαμβάνονται τα services, τα οποία μπορεί να χρησιμοποιήσει ο collector ως client, με τον τρόπο που περιγράψαμε παραπάνω. Όλα τα sensor services είναι προαιρετικά και στους δύο ρόλους. Υποχρεωτικό είναι μόνο το weather station service (master service), το οποίο πρέπει να υποστηρίζεται από όλες τις συμβατές συσκευές. Ορίζουμε τα sensor services ως προαιρετικά, γιατί, το ποια θα υπάρχουν στον weather station, εξαρτάται από τους αισθητήρες που αυτός περιλαμβάνει. Προφανώς, πρέπει να υπάρχει τουλάχιστον ένας αισθητήρας και το αντίστοιχο service, διαφορετικά το master service δεν έχει καμία λειτουργικότητα. Ο collector, από τη μεριά του, χρειάζεται να υποστηρίζει, εκτός του master service, μόνο τα services που αντιστοιχούν στα μεγέθη που τον ενδιαφέρουν. Μια μη συμβατή συσκευή, η οποία όμως υποστηρίζει κάποιο από τα sensor services, μπορεί να χρησιμοποιήσει μόνο το συγκεκριμένο service, χωρίς πρόβλημα.

Εκτός από το master και τα sensor services, ορίζουμε ως προαιρετικό το SIG Device Information Service, το οποίο παρέχει πληροφορίες σχετικά με την ίδια τη συσκευή καταγραφής, όπως ο κατασκευαστής της, το model number της και το system ID. Αυτές μπορούν, για παράδειγμα. να χρησιμοποιηθούν για τη διάκριση όμοιων συσκευών.

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile
    ...
    type="gr.ntua.microlab.profile.weather_station"
    name="Weather Station">
...
    <Role name="Weather Station">
        <Service type="gr.ntua.microlab.service.weather_station">
            <Declaration>Primary</Declaration>
            <Requirement>Mandatory</Requirement>
        </Service>

    ...
        <Service type="gr.ntua.microlab.service.temperature_sensor">
            <Declaration>Primary</Declaration>
            <Requirement>Optional</Requirement>
        </Service>
        <Service type="gr.ntua.microlab.service.humidity_sensor">
            <Declaration>Primary</Declaration>
            <Requirement>Optional</Requirement>
        </Service>
        <Service type="gr.ntua.microlab.service.barometric_pressure_sensor">
            <Declaration>Primary</Declaration>
            <Requirement>Optional</Requirement>
        </Service>
        <Service type="gr.ntua.microlab.service.ambient_light_sensor">
            <Declaration>Primary</Declaration>
            <Requirement>Optional</Requirement>
        </Service>
        <Service type="gr.ntua.microlab.service.uv_index_sensor">
            <Declaration>Primary</Declaration>
            <Requirement>Optional</Requirement>
        </Service>
    ...
```

```
        <Service type="org.bluetooth.service.device_information">
            <Declaration>PrimarySingleInstance</Declaration>
            <Requirement>Optional</Requirement>
            <Characteristic
               type="org.bluetooth.characteristic.manufacturer_name_string">
                <Requirement>Mandatory</Requirement>
            </Characteristic>
            <Characteristic
                type="org.bluetooth.characteristic.model_number_string">
                <Requirement>Mandatory</Requirement>
            </Characteristic>
            <Characteristic
                type="org.bluetooth.characteristic.system_id">
                <Requirement>Mandatory</Requirement>
            </Characteristic>
        </Service>
    </Role>

    <Role name="Weather Collector">
        <Client type="gr.ntua.microlab.service.weather_station">
            <Requirement>Mandatory</Requirement>
        </Client>

    ...
        <Client type="gr.ntua.microlab.service.temperature_sensor">
            <Requirement>Optional</Requirement>
        </Client>
        <Client type="gr.ntua.microlab.service.humidity_sensor">
            <Requirement>Optional</Requirement>
        </Client>
        <Client type="gr.ntua.microlab.service.barometric_pressure_sensor">
            <Requirement>Optional</Requirement>
        </Client>
        <Client type="gr.ntua.microlab.service.ambient_light_sensor">
            <Requirement>Optional</Requirement>
        </Client>
        <Client type="gr.ntua.microlab.service.uv_index_sensor">
            <Requirement>Optional</Requirement>
        </Client>
    ...

        <Client type="org.bluetooth.service.device_information">
            <Requirement>Optional</Requirement>
        </Client>
    </Role>
</Profile>
```

6.6 Ορισμός UUIDs

Εφόσον ορίσαμε ένα custom profile, το οποίο αποτελείται από custom services και custom characteristics, έπρεπε να ορίσουμε και τα αντίστοιχα 128-bit UUIDs για καθένα από αυτά. Για τον ορισμό των UUIDs, χρησιμοποιήσαμε την τεχνική που χρησιμοποιείται από τα SIG UUIDs. Δημιουργήσαμε, δηλαδή, ένα 128-bit UUID και, με βάση αυτό, ορίσαμε όλα τα υπόλοιπα, προσθέτοντας στο βασικό UUID, ένα, συγκεκριμένο για το καθένα, 16-bit αναγνωριστικό. Βέβαια, οι τιμές των UUIDs δεν έχουν σημασία για τη λειτουργία του προφίλ. Θα μπορούσαν οι τιμές τους να είναι τελείως διαφορετικές. Χρησιμοποιήσαμε την τεχνική αυτή, πρώτον για λόγους ευκολίας στον ορισμό των UUIDs, κυρίως όμως, όπως θα δούμε στο επόμενο κεφάλαιο, ώστε να εξοικονομήσουμε χώρο στο ενσωματωμένο σύστημα από την αποφυγή αποθήκευσης πολλών UUIDs. Το βασικό UUID είναι το ακόλουθο:

{DC980000-F292-11E3-B75F-002215F5EF22}

Το αναγνωριστικό κάθε, παραγόμενου από αυτό, UUID προστίθεται στο υπογραμμισμένο μέρος. Στον ακόλουθο πίνακα περιέχονται κάποια ενδεικτικά αναγνωριστικά.

Type	**UUID**
gr.ntua.microlab.service.weather_station	0x1000
gr.ntua.microlab.characteristic.weather_station_measurement	0x1001
gr.ntua.microlab.characteristic.weather_station_measurement_interval	0x1002
gr.ntua.microlab.characteristic.weather_station_control_point	0x1003
gr.ntua.microlab.service.temperature_sensor	0x1100
gr.ntua.microlab.characteristic.temperature_sensor_measurement	0x1101
gr.ntua.microlab.characteristic.temperature_sensor_measurement_interval	0x1102
gr.ntua.microlab.characteristic.temperature_sensor_control_point	0x1103
gr.ntua.microlab.service.humidity_sensor	0x1200
gr.ntua.microlab.characteristic.humidity_sensor_measurement	0x1201
gr.ntua.microlab.characteristic.humidity_sensor_measurement_interval	0x1202
gr.ntua.microlab.characteristic.humidity_sensor_control_point	0x1203

6.7 Συμπεριφορά συμβατών συσκευών

Σε αυτή την υπο-ενότητα περιγράφουμε επιπλέον συμπεριφορές και λειτουργίες που προβλέπονται για συσκευές, συμβατές με το weather station profile.

6.7.1 Εύρεση Weather Station

Ο weather station ξεκινά τη διαδικασία advertising, χρησιμοποιώντας το τύπο connectable undirected advertising, το οποίο σημαίνει ότι είναι ταυτόχρονα και scannable. Στα advertising data, περιλαμβάνεται το GAP name της συσκευής, το οποίο μπορεί, αν υποστηρίζεται από τη συσκευή, να μεταβληθεί από τον collector (η μεταβολή αυτή είναι ορατή σε όλες τις συσκευές). Στα scan response data, περιλαμβάνεται το UUID του weather station service, χρησιμοποιώντας τον τύπο “Incomplete List of 128-bit Service UUIDs”. Προκειμένου ο collector να ανακαλύψει τον weather station, ξεκινά τη διαδικασία active scanning. Για κάθε συσκευή που ανακαλύπτει, ελέγχει τα scan response data για το weather station service UUID. Σε περίπτωση που το UUID περιέχεται στα scan response data, η συγκεκριμένη συσκευή θεωρείται ότι είναι weather station, και ο collector μπορεί να ξεκινήσει τη διαδικασία σύνδεσης.

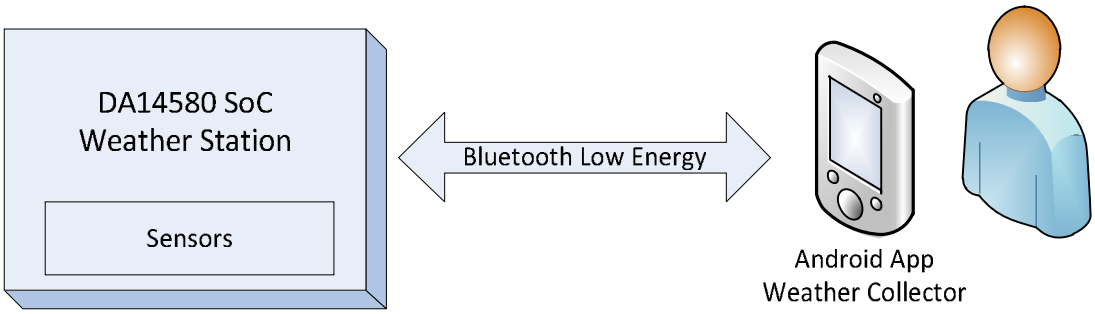
Weather Station Service UUID: {DC981000-F292-11E3-B75F-002215F5EF22}

6.7.2 Τιμές παραμέτρων

Κατά την πρώτη σύνδεση του weather station με έναν collector, οι παράμετροι λειτουργίας του συστήματος τίθενται σε default τιμές. Τα client characteristic configuration descriptors των measurement και control point τίθενται, σε όλα τα services, σε μηδενική τιμή, όπως ορίζει το πρότυπο του BLE, δηλαδή είναι απενεργοποιημένα. Επίσης, όλα τα measurement intervals τίθενται σε μηδενική τιμή, δηλαδή οι περιοδικές μετρήσεις είναι απενεργοποιημένες. Το γεγονός, ότι οι μετρήσεις και οι αποστολές του master service είναι απενεργοποιημένες, εξασφαλίζει ότι οι υπηρεσίες των αισθητήρων μπορούν να λειτουργούν αυτόνομα και να χρησιμοποιούνται από clients ανεξάρτητα από το master service. Αρχικά, οι μετρήσεις από το master service είναι απενεργοποιημένες για όλους τους αισθητήρες. Αντίθετα, η δυνατότητα αποστολής δεδομένων μέσω του master service είναι ενεργοποιημένη για όλους τους αισθητήρες. Μετά τη σύνδεση, ο collector μπορεί να γράψει το απαιτούμενο configuration στη συσκευή. Αν οι συσκευές είναι bonded, τότε, εκτός των client characteristic configuration descriptors, τα οποία διατηρούν τις τιμές τους μεταξύ συνδέσεων, όπως ορίζει το πρότυπο του BLE, το ίδιο ισχύει και για τις παραπάνω παραμέτρους. Δηλαδή, το configuration του συστήματος διατηρείται μεταξύ των συνδέσεων για bonded συσκευές. Διαφορετικά, πρέπει να γίνεται σε κάθε σύνδεση.

Κεφάλαιο 7

DA14580: Weather Station



Για την υλοποίηση του τμήματος εκείνου του συστήματος καταγραφής καιρού, το οποίο πραγματοποιεί τις μετρήσεις των σχετικών με τον καιρό δεδομένων, χρησιμοποιήσαμε, όπως έχουμε ήδη αναφέρει, το Bluetooth Low Energy System on a Chip DA14580, της εταιρίας Dialog Semiconductor, καθώς και την αντίστοιχη αναπτυξιακή πλακέτα και το SDK που παρέχονται, με σκοπό τον έλεγχο και την αποσφαλμάτωση BLE εφαρμογών, βασισμένων στο DA14580, πριν αυτές περάσουν στην παραγωγή. Η εφαρμογή που εκτελείται στο DA14580 SoC πραγματοποιεί τις απαραίτητες μετρήσεις, χρησιμοποιώντας μια σειρά από κατάλληλους αισθητήρες, και με βάση τις ρυθμίσεις λειτουργίας από τον χρήστη. Στη συνέχεια, στέλνει τα αποτελέσματά τους στο δεύτερο τμήμα, το οποίο αποτελεί το user interface του συστήματος καταγραφής καιρού. Το δεύτερο τμήμα του συστήματος εκτελείται σε κάποιο Android smartphone ή tablet και θα περιγραφεί στο επόμενο κεφάλαιο. Η επικοινωνία μεταξύ των δύο τμημάτων επιτυγχάνεται μέσω του πρωτοκόλλου BLE και εκτελείται με βάση το Weather Station Profile, το οποίο παρουσιάσαμε στο προηγούμενο κεφάλαιο. Το τελευταίο καθορίζει, όπως είδαμε, τους ρόλους που αντιστοιχούν σε κάθε συσκευή, τη μορφή των δεδομένων και των υπηρεσιών που αυτές πρέπει να υποστηρίζουν, και τη γενικότερη συμπεριφορά που αναμένεται από τις συμβατές συσκευές, προκειμένου να υλοποιήσουν τη συγκεκριμένη περίπτωση χρήσης.

Σε αυτό το κεφάλαιο περιγράφουμε τη δομή και τον τρόπο λειτουργίας της ενσωματωμένης εφαρμογής, η οποία εκτελείται στο DA14580 SoC. Αφού κάνουμε μια γενική περιγραφή του συστήματος, των βασικών συστατικών μερών από τα οποία αποτελείται και των τρόπων αλληλεπίδρασής τους, θα ακολουθήσει λεπτομερής περιγραφή κάθε συστατικού στοιχείου.

7.1 Γενική περιγραφή του συστήματος

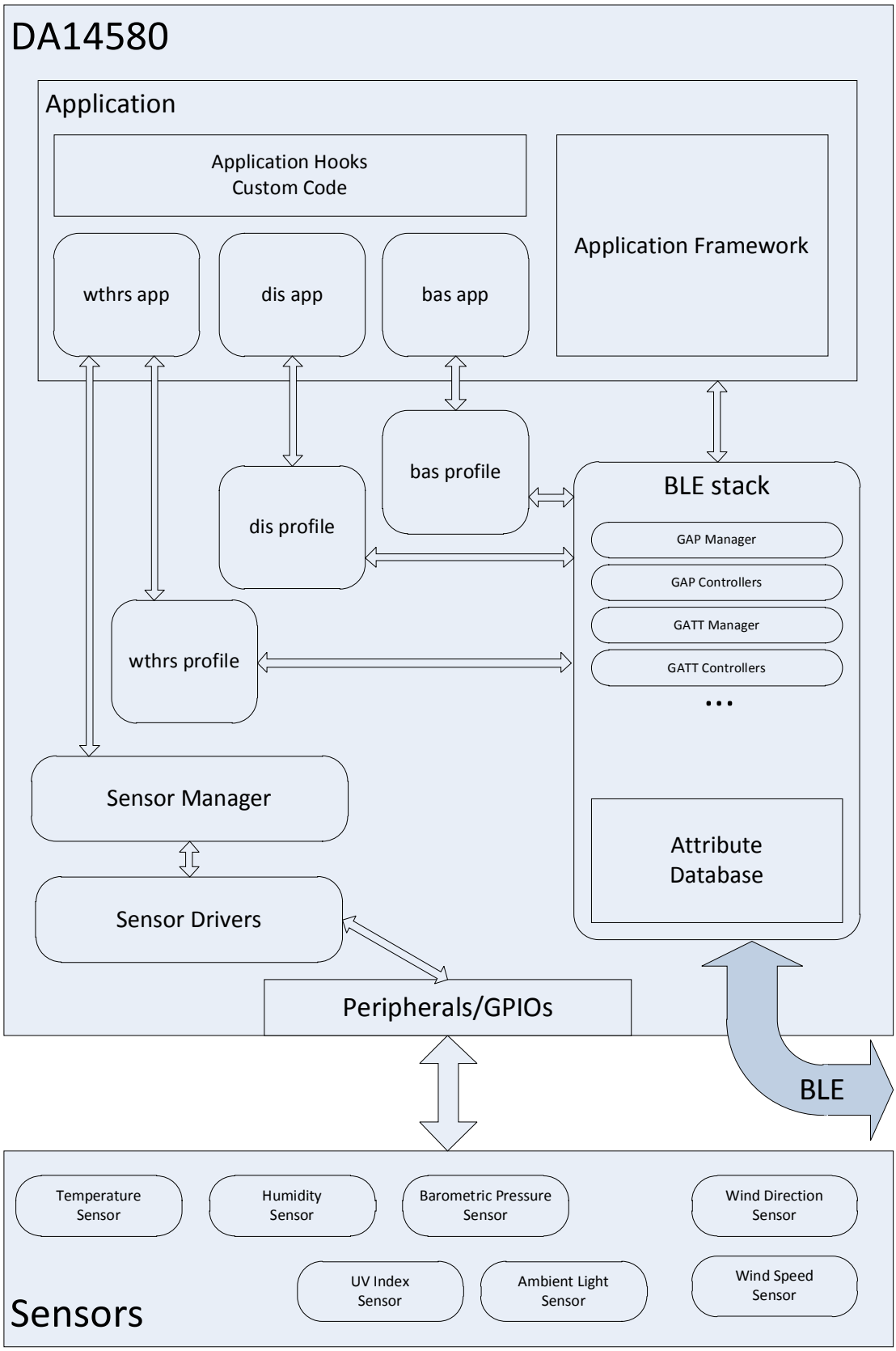
Έχουμε ήδη περιγράψει σε προηγούμενο κεφάλαιο τη δομή και τον τρόπο λειτουργίας του DA14580. Αναφέρουμε, επιγραμματικά, κάποια βασικά στοιχεία. Το DA14580 firmware περιέχει τον kernel, ένα μικρό και αποδοτικό real-time operating system, το οποίο παρέχει τη δυνατότητα δημιουργίας tasks και ανταλλαγής μηνυμάτων μεταξύ αυτών. Τα στοιχεία της στοίβας πρωτοκόλλων του BLE υλοποιούνται ως μια σειρά από managers και controllers, καθένας από τους οποίους έχει μηδέν ή περισσότερα tasks. Όλη η λειτουργία του συστήματος βασίζεται στην ανταλλαγή μηνυμάτων. Τα μηνύματα αυτά μπορεί να είναι εντολές για την εκτέλεση λειτουργιών από το task προορισμού, ή ενημερώσεις για γεγονότα που συμβαίνουν.

Στο SDK περιέχονται, επίσης, υλοποιήσεις πολλών από τα SIG adopted profiles με τη μορφή κώδικα, τα οποία μπορούν να ενεργοποιηθούν με κατάλληλα preprocessor macros και προσθήκη κώδικα σε συγκεκριμένα σημεία της εφαρμογής. Επίσης, το SDK παρέχει ένα application framework, πάνω στο οποίο ένας developer μπορεί να χτίσει την εφαρμογή του, υλοποιώντας μια σειρά από hook συναρτήσεις, που ορίζονται για τον σκοπό αυτό. Το framework πραγματοποιεί μεγάλο μέρος των απαραίτητων για τη λειτουργία του συστήματος αρχικοποιήσεων και υλοποιεί τη βασική λειτουργικότητα μιας BLE εφαρμογής. Όπου απαιτείται η υλοποίηση της συγκεκριμένης συμπεριφοράς που προβλέπεται για κάθε περίπτωση χρήσης, το framework καλεί τις αντίστοιχες hook συναρτήσεις, οι οποίες παρέχονται από την εφαρμογή.

Εξαιτίας της σπουδαιότητας των GATT profiles στη δημιουργία BLE εφαρμογών, το SDK παρέχει κατάλληλο API για τον χειρισμό των προφίλ, ενώ το application framework έχει οριστεί με βάση τη λειτουργία τους. Το application framework, πέρα από τις βασικές BLE διαδικασίες που εκκινεί, όπως advertising, scanning, connection, λειτουργεί κυρίως ως profile container, με βασική λειτουργία την αρχικοποίηση και ενεργοποίηση των προφίλ, τα οποία, στη συνέχεια, αναλαμβάνουν την πραγματοποίηση των λειτουργιών που προβλέπονται για το καθένα, από το αντίστοιχο πρότυπο. Όπως είδαμε, η υλοποίηση κάθε προφίλ χωρίζεται σε δύο τμήματα. Το πρώτο είναι ανεξάρτητο της εφαρμογής και υλοποιεί τη γενική λειτουργικότητα που προβλέπεται από το προφίλ, ενώ το δεύτερο είναι εξαρτώμενο από την εφαρμογή και υλοποιεί λειτουργίες που σχετίζονται με τη συγκεκριμένη περίπτωση χρήσης. Τα τμήματα αυτά υλοποιούνται ως tasks, με το δεύτερο να χρησιμοποιεί το task της εφαρμογής. Επικοινωνούν μεταξύ τους και με το BLE stack με τη χρήση κατάλληλων μηνυμάτων.

Τα βασικά συστατικά στοιχεία της εφαρμογής παρουσιάζονται στο διάγραμμα της διπλανής σελίδας. Ακολουθεί η συνοπτική περιγραφή τους.

Ένα από τα βασικότερα στοιχεία του συστήματος καταγραφής καιρού είναι, βέβαια, οι αισθητήρες, που εκτελούν τις μετρήσεις και παρέχουν τα καιρικά δεδομένα. Οι αισθητήρες συνδέονται στο DA14580 με τη βοήθεια των περιφερειακών και των GPIOs που παρέχονται από αυτό. Για κάθε αισθητήρα υπάρχει ο αντίστοιχος driver, ο οποίος υλοποιείται με βάση το specification του, το οποίο παρέχεται από την εταιρία κατασκευής του. Ο driver μπορεί να ορίζει λειτουργίες όπως αρχικοποίηση, εκκίνηση μέτρησης, διάβασμα αποτελεσμάτων, και είναι το interface μεταξύ της εφαρμογής και του αισθητήρα. Για τον κεντρικό χειρισμό των αισθητήρων και των λειτουργιών τους, δημιουργήσαμε ένα module, τον sensor manager. Ο sensor manager εκτελεί τις γενικές λειτουργίες που προβλέπονται για όλους τους αισθητήρες, καλώντας όπου χρειάζεται, διαφορετικές συναρτήσεις για κάθε έναν από αυτούς, για την πραγματοποίηση της διακριτής συμπεριφοράς του. Ο sensor manager ορίζει ένα task, το οποίο χρησιμοποιεί για την επικοινωνία του με τα υπόλοιπα modules του συστήματος.



Δομή της ενσωματωμένης εφαρμογής.

Για την υλοποίηση του Weather Station Profile, το οποίο περιγράψαμε στο προηγούμενο κεφάλαιο, ακολουθήσαμε την ίδια λογική που χρησιμοποιείται από τα υλοποιημένα στο SDK profiles. Χωρίσαμε, δηλαδή, την υλοποίηση σε δύο τμήματα. Το ανεξάρτητο της εφαρμογής τμήμα (wthrs profile) υλοποιεί λειτουργίες όπως η δημιουργία του master service και των sensor services στη βάση δεδομένων του ATT, η αρχικοποίησή τους κατά τη σύνδεση με τον collector, η αποστολή των αποτελεσμάτων μετρήσεων σε αυτόν, καθώς και ο έλεγχος των εγγραφών στη βάση δεδομένων, μέσω των οποίων μπορεί ο χρήστης του collector να ρυθμίσει τις παραμέτρους λειτουργίας του συστήματος. Το wthrs profile ορίζει ένα task και κατάλληλα μηνύματα για την επικοινωνία του με το άλλο τμήμα. Το εξαρτώμενο από την εφαρμογή τμήμα (wthrs app) είναι το interface μεταξύ της εφαρμογής, του wthrs profile και του sensor manager. Χρησιμοποιώντας τα μηνύματα που ορίζονται από αυτά, επικοινωνεί μαζί τους, προκειμένου να υλοποιήσει λειτουργίες όπως η εκτέλεση μετρήσεων και εντολών, που στέλνονται από τον collector, με τον τρόπο που ορίζεται στο προφίλ, η ενεργοποίηση του sensor manager κατά τη σύνδεση με τον collector, και η αποστολή στο wthrs profile αποτελεσμάτων μετρήσεων, μετά από ενημέρωση από τον sensor manager, προκειμένου αυτά να σταλούν στον collector μέσω GATT notification.

Χρησιμοποιήσαμε επίσης, το παρεχόμενο από το SDK, Device Information Service, το οποίο ορίζεται ως προαιρετικό στο Weather Station Profile. Όπως αναφέραμε στη σχετική υπο-ενότητα, το Device Information Service (DIS) παρέχει πληροφορίες για τη συσκευή και μπορεί, για παράδειγμα, να χρησιμοποιηθεί για τη διάκριση όμοιων συσκευών. Ένα άλλο service, που παρέχεται από το SDK και το οποίο ενεργοποιήσαμε, είναι το Battery Service (BAS). Αυτό παρέχει πληροφορίες σχετικά με το επίπεδο της μπαταρίας της συσκευής. Αν και δεν προβλέπεται από το Weather Station Profile, είναι ένα service που περιλαμβάνεται συνήθως σε Bluetooth Smart συσκευές, οι οποίες είναι προορισμένες να λειτουργούν με μπαταρία. Με τη βοήθειά του, ο χρήστης μπορεί να πληροφορηθεί αν είναι απαραίτητη η αλλαγή μπαταρίας στη συσκευή. Το DA14580 παρέχει αυτές τις πληροφορίες μέσω του ADC περιφερειακού και κατάλληλου driver.

Η συσκευή καταγραφής καιρού και η ενσωματωμένη εφαρμογή που εκτελείται σε αυτή εμφανίζονται με διάφορους τρόπους και ρόλους στα διαφορετικά επίπεδα της αρχιτεκτονικής του συστήματος. Πιο συγκεκριμένα, έχουμε:

- **GAP**: Στο επίπεδο του GAP, η συσκευή καταγραφής είναι peripheral.
- **GATT**: Στο επίπεδο του GATT, η συσκευή καταγραφής είναι GATT server.
- **ATT**: Στο επίπεδο του ATT, ισχύει ό,τι και στο επίπεδο του GATT, οπότε η συσκευή καταγραφής είναι ATT server.
- **Link Layer**: Στο Link Layer, η συσκευή καταγραφής λειτουργεί ως advertiser και, όταν δημιουργηθεί σύνδεση με κάποια απομακρυσμένη συσκευή, μετέχει στη σύνδεση με το ρόλο του slave.
- **Weather Station Profile**: Στο weather station profile, η συσκευή καταγραφής έχει το ρόλο του weather station.
- **Τύπος Αρχιτεκτονικής**: Ο τύπος της αρχιτεκτονικής του DA14580, που χρησιμοποιήθηκε για την υλοποίηση της εφαρμογής, είναι η αρχιτεκτονική fully hosted. Δηλαδή, η εφαρμογή εκτελείται μαζί με όλα τα υπόλοιπα στοιχεία του συστήματος στον κεντρικό επεξεργαστή του.
- **Τύπος Συσκευής**: Ο τύπος της συσκευής, με βάση το διαχωρισμό που έχει ορίσει το SIG, είναι Bluetooth Smart. Δηλαδή single-mode συσκευή, που παρέχει συγκεκριμένη λειτουργικότητα μέσω του GATT.

7.2 Γενικά / Βοηθητικά στοιχεία

Σε αυτή την υπο-ενότητα περιγράφουμε κάποια γενικά στοιχεία της εφαρμογής, καθώς και βοηθητικά στοιχεία που δημιουργήσαμε για τον έλεγχο της λειτουργίας της.

7.2.1 Application Template

Προκειμένου να διευκολύνει την ανάπτυξη εφαρμογών, το SDK του DA14580 παρέχει ένα template μVision project για BLE εφαρμογές. Το template δεν έχει καμία λειτουργικότητα, όμως, εκτός του ότι περιέχει τις απαραίτητες ρυθμίσεις, ώστε η εφαρμογή να μπορεί να φορτωθεί στην αναπτυξιακή πλακέτα, παρέχει επίσης όλα τα application hooks ως stubs. Σε αυτά μπορεί, με τη σειρά του, ο developer να προσθέσει τον απαραίτητο για τη δική του εφαρμογή κώδικα. Ως βάση, λοιπόν, για την υλοποίηση της ενσωματωμένης εφαρμογής, χρησιμοποιήσαμε το template project, εφαρμόζοντας κατάλληλες αλλαγές, όπως περιγράφεται στο documentation του SDK.

7.2.2 Application Configuration

Για την κεντρική ρύθμιση των παραμέτρων λειτουργίας της εφαρμογής στο στάδιο της μεταγλώττισης, υπάρχει ένα header file (da14580_config.h), το οποίο περιέχει μια σειρά από preprocessor macros της C, και γίνεται include σε όλα τα αρχεία. Με τη βοήθεια αυτών, μπορούν να ενεργοποιηθούν ή αν απενεργοποιηθούν συγκεκριμένα χαρακτηριστικά της εφαρμογής ή να γίνουν επιλογές, που καθορίζουν τον τρόπο λειτουργίας της. Η δυνατότητα απενεργοποίησης features κατά το compilation έχει σημασία σε embedded systems, γιατί μειώνει το μέγεθος του κώδικα, άρα και τη μνήμη που απαιτείται για την αποθήκευσή του. Για παράδειγμα, στο DA14580, το μέγιστο μέγεθος του κώδικα και των δεδομένων μιας εφαρμογής είναι όσο το μέγεθος της OTP μνήμης, δηλαδή 32KB. Για τον λόγο αυτό, ορίσαμε επιπλέον macros, τα οποία δίνουν τη δυνατότητα απενεργοποίησης συγκεκριμένων χαρακτηριστικών που υλοποιήσαμε, αν αυτά δεν είναι απαραίτητα.

Μια βασική ρύθμιση που γίνεται στο configuration header file, είναι η επιλογή του τύπου της αρχιτεκτονικής. Ορίζοντας το κατάλληλο macro, επιλέγουμε την αρχιτεκτονική fully hosted.

```
#define CFG_APP
```

Άλλες σημαντικές ρυθμίσεις, που μπορούν να γίνουν εδώ, είναι η ενεργοποίηση των sleep modes, ο καθορισμός παραμέτρων του memory map, η ενεργοποίηση των χαρακτηριστικών ασφάλειας του BLE και ο αριθμός των ταυτόχρονων συνδέσεων που μπορούν να υπάρχουν. Η εφαρμογή που υλοποιήσαμε ελέγχθηκε και με ενεργοποίηση του Extended Sleep Mode. Το Deep Sleep Mode δεν μπορεί να χρησιμοποιηθεί στο στάδιο του development, γιατί απαιτεί την εγγραφή της εφαρμογής στην OTP μνήμη, μια διαδικασία μη αναστρέψιμη, οπότε μπορεί να γίνει μόνο στο τελικό στάδιο ελέγχου, αν αυτό κριθεί απαραίτητο. Η συσκευή καταγραφής καιρού μπορεί να έχει μόνο μία σύνδεση ενεργή κάθε χρονική στιγμή, όπως προβλέπεται στην έκδοση 4.0 του προτύπου του BLE για peripheral συσκευές. Όπως αναφέραμε στο σχετικό κεφάλαιο, ο συγκεκριμένος περιορισμός έχει αφαιρεθεί από το πρότυπο στην έκδοση 4.1, αν και στις περισσότερες περιπτώσεις χρήσης δεν υπάρχει λόγος ή δεν είναι εφικτό να υπάρχουν ταυτόχρονα πολλές συνδέσεις σε ένα peripheral. Στην προκειμένη περίπτωση, η υποστήριξη πολλών ταυτόχρονων συνδέσεων από τη συσκευή καταγραφής, θα αύξανε την πολυπλοκότητα και το μέγεθος του κώδικα, χωρίς να προσφέρει επιπλέον χρηστικότητα στο σύστημα.

Μέσω του configuration header file, γίνεται και η ενεργοποίηση των προφίλ. Ενεργοποιήσαμε, λοιπόν, τα προφίλ που συμπεριλαμβάνονται στην εφαρμογή, προσθέτοντας παράλληλα νέα macros για το weather station profile.

```
// =========== Used Profiles ===========

#define CFG_PRF_BASS // Battery Service
#define CFG_PRF_DISS // Device Information Service

// ========== Custom Profiles ==========

#define CFG_PRF_WTHRS // Weather Station Profile
```

7.2.3 Battery Service – Device Information Service

Τα δύο προφίλ που παρέχονται από το SDK, και τα οποία ενεργοποιήσαμε στην εφαρμογή μας, ακολουθούν τον τρόπο σχεδιασμού που ισχύει για όλα τα προφίλ του SDK, δηλαδή χωρίζονται σε δύο τμήματα, όπως φαίνεται και στο διάγραμμα του συστήματος, τα οποία λειτουργούν με τον τρόπο που έχει ήδη αναφερθεί. Για τη χρήση τους, απαιτείται απλώς η προσθήκη κώδικα σε κατάλληλα σημεία των application hooks. Οι συγκεκριμένες υπηρεσίες είναι σχετικά απλές, δεν απαιτούν ιδιαίτερο configuration, και, ως εκ τούτου, μετά την ενεργοποίησή τους, λειτουργούν με την αναμενόμενη συμπεριφορά, ανεξάρτητα από το υπόλοιπο σύστημα. Για αυτό τον λόγο δε θα προχωρήσουμε σε αναλυτική περιγραφή τους.

7.2.4 Debug Log

Για τη διαδικασία ελέγχου και αποσφαλμάτωσης της εφαρμογής, υλοποιήσαμε μια σειρά από βοηθητικά macros και συναρτήσεις, τα οποία δίνουν τη δυνατότητα δημιουργίας debug log από την εφαρμογή. Για την υλοποίηση, χρησιμοποιήθηκε το UART περιφερειακό του DA14580, το οποίο δίνει τη δυνατότητα σειριακής σύνδεσης με έναν υπολογιστή. Έτσι, χρησιμοποιώντας κατάλληλο λογισμικό σειριακής σύνδεσης (π.χ. putty), μπορούμε να βλέπουμε το debug log που παράγεται κατά την εκτέλεση της εφαρμογής. Η υλοποίηση του debug log έγινε με τέτοιο τρόπο, ώστε να μπορεί να απενεργοποιείται, τόσο κεντρικά, όσο και για κάθε module ξεχωριστά, με τη χρήση κατάλληλων macros. Έτσι, αν δε χρειάζεται, μπορεί να αφαιρεθεί, μειώνοντας το μέγεθος του κώδικα και τον χρόνο εκτέλεσης.

Το debug log υπήρξε ιδιαίτερα χρήσιμο στον έλεγχο της ενσωματωμένης εφαρμογής, ειδικά σε ό,τι αφορούσε τη γενικότερη συμπεριφορά της και την ανακάλυψη προβλημάτων που δεν οφείλονταν σε λάθη του κώδικα, αλλά είχαν να κάνουν με τη λογική του προγράμματος. Επίσης, πρέπει να αναφέρουμε ότι η διαδικασία αποσφαλμάτωσης σε real-time συστήματα καθίσταται δυσκολότερη εξαιτίας του τρόπου λειτουργίας τους. Για παράδειγμα, αν η εφαρμογή παρουσιάζει μη αναμενόμενη συμπεριφορά κατά τη σύνδεση, είναι δύσκολο να γίνει έλεγχος με τους συνηθισμένους τρόπους debugging (π.χ. breakpoints και έλεγχος των μεταβλητών), διότι η συσκευή με την οποία έχει γίνει η σύνδεση δε θα περιμένει και θα αποσυνδεθεί, όταν εκπνεύσει το supervision timeout.

Ακολουθεί ενδεικτικό τμήμα του debug log κατά την εκκίνηση της εφαρμογής, από τις αρχικοποιήσεις και τη δημιουργία του database μέχρι την αρχή του advertising. Ένα επιπλέον γεγονός που ελήφθη υπόψη, το οποίο φαίνεται και στο ακόλουθο log, είναι ότι όλα τα debug strings περιέχονται στον κώδικα της εφαρμογής και, αναγκαστικά, πρέπει να διατηρούνται όσο το δυνατόν μικρότερα.

```
[WTHRS] ini
[SNSMGR] ini
[WTHRS_PROJ] ini
[WTHRS_APP] ini
[WTHRS_APP] mgr ini
[WTHRS_PROJ] sec ini
[SNSMGR_TASK] ini
[WTHRS] ini
[SNSMGR] ini
[WTHRS_PROJ] cfg
[WTHRS_PROJ] cfg cmp
[WTHRS_PROJ] DB init
[WTHRS_PROJ] DB init
[WTHRS_APP] DB
[WTHRS_TASK] DB
WTH
[ATTMEXT] CREATE
s=1000(0008) c=3
SIZE:114 u16=9 u128=3
SVC:0010
INC:0,3
CHR:1001(0008) d=0
DSC: cc
CHR:1002(0008) d=0
CHR:1003(0008) d=0
DSC: cc
WTH=0
TMP
[ATTMEXT] CREATE
s=1100(0008) c=3
SIZE:91 u16=7 u128=3
SVC:001c
INC:0,0
CHR:1101(0008) d=0
DSC: cc pf
CHR:1102(0008) d=0
CHR:1103(0008) d=0
DSC: cc
TMP=0
[ATTMEXT] CREATE
s=1200(0008) c=3
SIZE:91 u16=7 u128=3
SVC:0026
INC:0,0
CHR:1201(0008) d=0
DSC: cc pf
CHR:1202(0008) d=0
CHR:1203(0008) d=0
DSC: cc
HMD=0
...
[ATTMEXT] SETINC:1000(0008)
[WTHRS_APP_TASK] DBcf:0
[WTHRS_PROJ] DB init
[WTHRS_APP] mgr str
[SNSMGR_TASK] STR:0,00ff
...
[WTHRS_PROJ] DB init
[WTHRS_PROJ] DB cmp
[WTHRS_PROJ] adv
...
```

7.3 Application Hooks

Για τη σύνδεση της εφαρμογής με το application framework, που παρέχεται από το SDK του DA14580, υλοποιήσαμε τα απαραίτητα για την εφαρμογή μας application hooks. Αναφέραμε παραπάνω ότι, πέρα από κάποιες βασικές λειτουργίες του BLE, όπως σύνδεση ή αποσύνδεση, που είναι ανεξάρτητες των profiles και εκτελούνται από το application framework, αυτό λειτουργεί περισσότερο ως profile container. Αυτό συμβαίνει επειδή η λογική της εφαρμογής ορίζεται στα profiles και υλοποιείται από τον αντίστοιχο κώδικα. Οπότε το framework, το μόνο που έχει να κάνει, είναι να αρχικοποιήσει τα απαιτούμενα προφίλ, τα οποία αναλαμβάνουν από εκεί και πέρα την υλοποίηση της συγκεκριμένης περίπτωσης χρήσης. Περιγράφουμε, εδώ, τις βασικές hook συναρτήσεις που υλοποιήσαμε.

7.3.1 periph_init

Η periph_init καλείται κατά την εκκίνηση του συστήματος και μετά την έξοδο από sleep mode. Σε αυτή τη συνάρτηση, αρχικοποιούμε τα απαραίτητα περιφερειακά. Το βασικότερο είναι το I^2C bus, που χρησιμοποιείται για τη σύνδεση των περισσότερων αισθητήρων. Επίσης αν το debug log είναι ενεργοποιημένο, αρχικοποιείται και το UART.

```
// ============================== UART ==============================
#ifdef PROGRAM_ENABLE_UART
GPIO_ConfigurePin(GPIO_PORT_0, GPIO_PIN_4, OUTPUT, PID_UART1_TX, false);
#ifndef CFG_UART_DISABLE_RX
GPIO_ConfigurePin(GPIO_PORT_0, GPIO_PIN_5, INPUT, PID_UART1_RX, false);
#endif
...
#endif // PROGRAM_ENABLE_UART

...
SetBits16(CLK_PER_REG, UART1_ENABLE, 1); // enable clock
...
uart_init(UART_BAUDRATE_115K2, 3);
...

// ============================== I2C ==============================
GPIO_SetPinFunction(GPIO_I2C_SDA_PORT, GPIO_I2C_SDA_PIN,
                    INPUT, PID_I2C_SCL);
GPIO_SetPinFunction(GPIO_I2C_SCL_PORT, GPIO_I2C_SCL_PIN,
                    INPUT, PID_I2C_SDA);
```

7.3.2 app_init_func

Η app_init_func καλείται πριν την είσοδο στο main loop. Σε αυτή τη συνάρτηση αρχικοποιούμε τα προφίλ και τον sensor manager.

```
void app_init_func(void)
{
    APP_LOG_TAG_STR(WTHRS_PROJ, "ini");

    #if (BLE_WEATHER_STATION)
    app_wthrs_init();
    #endif

    #if (BLE_SENSOR_MANAGER)
    app_snsmgr_init();
    #endif
```

```
    #if (BLE_BATT_SERVER)
    app_batt_init();
    #endif

    #if (BLE_DIS_SERVER)
    app_dis_init();
    #endif
}
```

7.3.3 app_configuration_func – app_set_dev_config_complete_func

Η app_configuration_func καλείται, προκειμένου η εφαρμογή να θέσει τις ρυθμίσεις του GAP manager. Σε αυτή τη συνάρτηση, θέτουμε το ρόλο της συσκευής σε peripheral (slave), ενώ μπορούμε να επιτρέψουμε την εγγραφή, από τον collector, του GAP name της συσκευής. Άλλες ρυθμίσεις που μπορούν να τεθούν είναι το MTU και οι τιμές των χαρακτηριστικών του GAP service: Appearance, Peripheral Privacy Flag, Peripheral Preferred Connection Parameters.

```
void app_configuration_func(ke_task_id_t const task_id,
                            struct gapm_set_dev_config_cmd *cmd)
{
    APP_LOG_TAG_STR(WTHRS_PROJ, "cfg");

    // set device configuration
    cmd->operation = GAPM_SET_DEV_CONFIG;
    // Device Role
    cmd->role = GAP_PERIPHERAL_SLV;
...
    // Device Name write permission requirements for peer device
    cmd->name_write_perm = GAPM_WRITE_DISABLE;
...
}
```

Μετά την ολοκλήρωση της ρύθμισης και την απάντηση από τον GAP manager, καλείται η app_set_dev_config_complete_func. Σε αυτή τη συνάρτηση ξεκινάμε τη διαδικασία αρχικοποίησης του ATT database.

```
void app_set_dev_config_complete_func(void)
{
...
    // We are now in Initialization State
    ke_state_set(TASK_APP, APP_DB_INIT);

    // Add the first required service in the database
    if (app_db_init())
    {
        // No service to add in the DB -> Start Advertising
        app_adv_start();
    }
...
}
```

7.3.4 app_db_init_func – app_db_init_complete_func

Η app_db_init_func καλείται για την αρχικοποίηση του ATT database. Η αρχικοποίηση συμβαίνει σειριακά. Αφού σταλεί σε κάθε προφίλ το κατάλληλο μήνυμα, αυτό δημιουργεί τις υπηρεσίες του στο ATT database και, με την ολοκλήρωση της διαδικασίας, στέλνει απάντηση στην εφαρμογή. Με τη λήψη της απάντησης η εφαρμογή προχωρά στο

επόμενο προφίλ μέχρι να ολοκληρωθεί η διαδικασία για όλα τα προφίλ. Παρόλο που ο sensor manager δεν αποτελεί προφίλ και δε δημιουργεί υπηρεσίες, καλούμε εδώ τη συνάρτηση εκκίνησης των αισθητήρων, ώστε αυτοί να είναι διαθέσιμοι πριν ξεκινήσει το advertising. Έτσι, εκμεταλλευόμαστε την ήδη υπάρχουσα λογική του framework, χωρίς να χρειάζεται να προσθέσουμε επιπλέον καταστάσεις και συμπεριφορά.

```
bool app_db_init_func(void)
{
    APP_LOG_TAG_STR(WTHRS_PROJ, "DB init");
...
    // Indicate if more services need to be added in the database
    bool end_db_create = false;
...
    // Check if another should be added in the database
    if (app_env.next_prf_init < APP_PRF_LIST_STOP)
    {
        switch (app_env.next_prf_init)
        {
            #if (BLE_WEATHER_STATION)
            case (APP_WTHRS_TASK):
            {
                app_wthrs_create_db_send();
            } break;
            #endif //BLE_WEATHER_STATION
            #if (BLE_SENSOR_MANAGER)
            case (APP_SNSMGR_TASK):
            {
                app_snsmgr_start();
            } break;
            #endif //BLE_SENSOR_MANAGER
            #if (BLE_BATT_SERVER)
            case (APP_BASS_TASK):
            {
                app_batt_create_db();
            } break;
            #endif //BLE_BATT_SERVER
            #if (BLE_DIS_SERVER)
            case (APP_DIS_TASK):
            {
                app_dis_create_db_send();
            } break;
            #endif //BLE_DIS_SERVER
            default:
            {
                ASSERT_ERR(0);
            } break;

        }

        // Select following service to add
        app_env.next_prf_init++;
    }
    else
    {
        end_db_create = true;
    }

    return end_db_create;
}
```

Με την ολοκλήρωση της παραπάνω διαδικασίας για όλα τα προφίλ, καλείται η app_db_init_complete_func. Σε αυτή τη συνάρτηση, εκκινούμε τη διαδικασία του advertising.

```
void app_db_init_complete_func(void)
{
    APP_LOG_TAG_STR(WTHRS_PROJ, "DB cmp");

    app_adv_start();
}
```

7.3.5 app_adv_func

Η app_adv_func καλείται, προκειμένου η εφαρμογή να θέσει τις παραμέτρους του advertising, καθώς και τα advertising και scan response data. Σε αυτή τη συνάρτηση, θέτουμε τον τύπο του advertising σε connectable undirected advertising, και τη συσκευή σε connectable και discoverable GAP modes. Αυτό σημαίνει ότι μπορεί να δεχτεί τόσο connection requests όσο και scan requests. Τα τελευταία είναι απαραίτητα για τη λειτουργία του weather station profile, αφού, μέσω των scan response data, γίνεται η αναγνώριση του weather station από τον collector. Προσθέτουμε, λοιπόν, το GAP name της συσκευής στα advertising data και το UUID του weather station service στα scan response data.

app_wthrs_proj.h

```
#define APP_SCNRSP_DATA \
  "\x11\x06\x22\xef\xf5\x15\x22\x00\x5f\xb7\xe3\x11\x92\xf2\x00\x10\x98\xdc"
#define APP_SCNRSP_DATA_LENGTH  (18)
```

app_wthrs_proj.c

```
void app_adv_func(struct gapm_start_advertise_cmd *cmd)
{
    APP_LOG_TAG_STR(WTHRS_PROJ, "adv");
...
    cmd->op.code     = GAPM_ADV_UNDIRECT;
    cmd->op.addr_src = GAPM_PUBLIC_ADDR;
    cmd->intv_min    = APP_ADV_INT_MIN;
    cmd->intv_max    = APP_ADV_INT_MAX;
...
    cmd->info.host.mode = GAP_GEN_DISCOVERABLE;
...
}
```

7.3.6 app_connection_func – app_disconnect_func

Η app_connection_func καλείται, προκειμένου η εφαρμογή να εκτελέσει τις απαραίτητες για αυτή διαδικασίες, κατά τη σύνδεση με μια απομακρυσμένη συσκευή. Σε αυτή τη συνάρτηση, ενημερώνουμε τα ενεργοποιημένα προφίλ για τη σύνδεση, ώστε να ενεργοποιήσουν τις υπηρεσίες τους. Επίσης, ενημερώνουμε τον sensor manager και εκτελούμε απαραίτητες λειτουργίες, όπως, για παράδειγμα, η ενεργοποίηση της περιοδικής μέτρησης του επιπέδου της μπαταρίας.

```
void app_connection_func(struct gapc_connection_req_ind const *param)
{
...
        #if (BLE_WEATHER_STATION)
```

```
        app_wthrs_enable();
        #endif

        #if (BLE_SENSOR_MANAGER)
        app_snsmgr_connection();
        #endif

        #if (BLE_BATT_SERVER)
        app_batt_enable(cur_batt_level, GPIO_BATT_LED_ALERT,
                        GPIO_BATT_LED_PORT, GPIO_BATT_LED_PIN);
        app_batt_poll_start(3000); // 5 mins (max)
        #endif

        #if (BLE_DIS_SERVER)
        app_dis_enable_prf(app_env.conhdl);
        #endif

        ke_state_set(TASK_APP, APP_CONNECTED);

        // Retrieve the connection info from the parameters
        app_env.conhdl = param->conhdl;
...
}
```

Η app_disconnect_func καλείται, προκειμένου η εφαρμογή να εκτελέσει τις απαραίτητες για αυτή διαδικασίες, κατά την αποσύνδεση από την απομακρυσμένη συσκευή. Η ενημέρωση για το γεγονός της αποσύνδεσης γίνεται με αποστολή, από τον GAP Controller της σύνδεσης, ενός μηνύματος τύπου GAPC_DISCONNECT_IND. Επειδή το συγκεκριμένο γεγονός είναι πολύ σημαντικό για τη λειτουργία όλων των στοιχείων της εφαρμογής, το μήνυμα αυτό στέλνεται σε κάθε task το οποίο είναι ενεργό. Έτσι, τα tasks των προφίλ και του sensor manager λαμβάνουν επίσης το συγκεκριμένο μήνυμα, και εκτελούν τις απαραίτητες για αυτά λειτουργίες αποσύνδεσης, στους αντίστοιχους handlers. Η εφαρμογή, λοιπόν, δε χρειάζεται να ενημερώσει τα προφίλ, αλλά απλώς να εκτελέσει τις επιπλέον διαδικασίες που μπορεί να απαιτούνται.

```
void app_disconnect_func(ke_task_id_t task_id,
                         struct gapc_disconnect_ind const *param)
{
...
    #if BLE_BATT_SERVER
    app_batt_poll_stop();
    #endif
...
        // Restart Advertising
        app_adv_start();
...
}
```

7.3.7 Sleep mode hooks

Η εφαρμογή που εκτελείται στον weather station δεν απαιτεί κάποια ιδιαίτερη επεξεργασία σε ό,τι αφορά τις διαδικασίες εισόδου σε sleep mode. Επίσης, δεν απαιτείται έλεγχος για ασύγχρονα γεγονότα, προκειμένου να συγχρονιστούν με την εφαρμογή ή να εκτελεστεί επιπλέον επεξεργασία. Έτσι, δεν είναι απαραίτητη η υλοποίηση των hook συναρτήσεων που ορίζονται στο SDK και καλούνται από το main loop για τον σκοπό αυτό.

7.4 Ορισμός προφίλ και υπηρεσιών

7.4.1 Υπηρεσίες

Προκειμένου η εφαρμογή να ενεργοποιήσει τις υπηρεσίες που απαρτίζουν κάθε προφίλ και να χρησιμοποιήσει τη λειτουργικότητα που αυτές ορίζουν, πρέπει πρώτα να τις δημιουργήσει στη βάση δεδομένων του Attribute Protocol. Όπως είδαμε κατά την περιγραφή του GATT, κάθε υπηρεσία ορίζεται, μέσω ενός service definition, ως μια ομάδα από attributes στο ATT database. Το service definition ξεκινά με το service declaration attribute, το οποίο ακολουθείται από μηδέν ή περισσότερα include definitions, ένα για κάθε service που γίνεται include από το αρχικό. Κάθε include definition αποτελείται από ακριβώς ένα include declaration attribute. Μετά τα include definitions, ακολουθούν τα characteristic definitions για τα χαρακτηριστικά της συγκεκριμένης υπηρεσίας. Καθένα από αυτά, ξεκινά με ακριβώς ένα characteristic declaration attribute, το οποίο ακολουθείται από ακριβώς ένα characteristic value declaration attribute και μηδέν ή περισσότερα characteristic descriptor declaration attributes, ένα για κάθε descriptor που ορίζεται για το χαρακτηριστικό.

Στο SDK του DA14580, ο ορισμός των υπηρεσιών από την εφαρμογή γίνεται απευθείας στο επίπεδο του ATT. Δηλαδή, αφού η εφαρμογή προσθέσει το service στη λίστα των services που διατηρείται από το ATT, ορίζοντας παράλληλα και το υπεύθυνο task για το συγκεκριμένο service, στη συνέχεια, δημιουργεί στο ATT database τα attributes που αναφέρθηκαν παραπάνω, με τη σειρά και τις τιμές που προβλέπονται από το πρότυπο του BLE. Για τη δημιουργία των attributes παρέχεται από το SDK κατάλληλο API. Επίσης, υπάρχει μία βοηθητική συνάρτηση για τη δημιουργία των attributes, η οποία δέχεται ως όρισμα ένα πίνακα από attributes και τα τοποθετεί στο ATT database, προσθέτοντας επιπλέον τιμές, όπου χρειάζεται. Αυτή, όμως, η συνάρτηση λειτουργεί μόνο για 16-bit UUIDs, οπότε δεν μπορεί να χρησιμοποιηθεί από τη δική μας εφαρμογή, η οποία χρησιμοποιεί 128-bit UUIDs.

Προκειμένου να αποφύγουμε τον ορισμό των υπηρεσιών μέσω κώδικα και το επιπρόσθετο κόστος αποθήκευσης των 128-bit UUIDs για κάθε attribute, αποφασίσαμε να υλοποιήσουμε μια βοηθητική συνάρτηση, παρόμοια με αυτή που παρέχεται από το SDK, η οποία, όμως, δέχεται ως όρισμα, όχι πλέον ένα πίνακα από attributes, αλλά μία δομή που αντιστοιχεί απευθείας στο service definition. Η δομή αυτή περιλαμβάνει βασικές πληροφορίες για το service, καθώς και έναν πίνακα από δομές, που αντιστοιχούν στα characteristic definitions των χαρακτηριστικών του service. Τα τελευταία μπορούν να περιέχουν επιπλέον πληροφορίες για τους descriptors του κάθε χαρακτηριστικού. Η συνάρτηση αυτή, χρησιμοποιώντας τις παραπάνω δομές, υπολογίζει το συνολικό χώρο που χρειάζεται για την αποθήκευση του service, και, στη συνέχεια, προσθέτει το service και δημιουργεί τα απαραίτητα attributes στο ATT database. Έτσι, αντί να ορίζουμε κάθε attribute ξεχωριστά, ορίζουμε απευθείας τα αντικείμενα του GATT, και η βοηθητική συνάρτηση τα μεταφράζει σε ένα ή περισσότερα attributes. Οι δομές για τα service και characteristic definitions είναι οι ακόλουθες:

```
// Characteristic definition
struct attm_ext_char_def
{
    // Characteristic properties
    uint8_t prop;
    // Characteristic UUID type
    uint8_t uuid_type;
    // Characteristic UUID (16-bit)
    uint16_t uuid;
    // Value max length
    att_size_t max_length;
    // Value initial length
```

```
    att_size_t length;
    // Value initial data (optional)
    uint8_t *value;
    // Value permissions
    uint16_t perm;
    // Characteristic descriptor flags
    uint8_t desc_flags;
    // Number of additional descriptors
    uint8_t nb_desc;
    // Characteristic descriptor definition array
    // (size = nb_desc + <value dependent on flags>)
    // Descriptor definition is absent only for client
    // configuration with default permissions.
    const union attm_ext_desc_def *char_desc;
};

// Service definition
struct attm_ext_svc_def
{
    // Service type
    uint8_t primary;
    // Number of included services with 16-bit UUIDs
    uint8_t nb_inc_svc_16;
    // Number of included services with 128-bit UUIDs
    uint8_t nb_inc_svc_128;
    // Number of characteristics
    uint8_t nb_char;
    // Service permissions
    uint8_t perm;
    // Service UUID type
    uint8_t uuid_type;
    // Service UUID (16-bit)
    uint16_t uuid;
    // Service characteristic definition array (size = nb_char)
    const struct attm_ext_char_def *svc_char;
};
```

Για τον ορισμό των UUIDs κάθε χαρακτηριστικού και υπηρεσίας, η εφαρμογή παρέχει έναν πίνακα με UUIDs, από τα οποία επιλέγει ένα, μέσω των πεδίων uuid_type και uuid των παραπάνω δομών. Υπάρχουν δύο περιπτώσεις, που ορίζονται με βάση την τιμή του uuid_type. Στην πρώτη περίπτωση, έχουμε UUIDs με κοινή βάση, όπως αυτά του SIG. Τα τέσσερα LSB του uuid_type χρησιμοποιούνται ως index στον πίνακα των UUIDs, για την επιλογή του βασικού UUID, ενώ το πεδίο uuid αποτελεί την τιμή που προστίθεται σε αυτό. Στη δεύτερη περίπτωση, το πεδίο uuid χρησιμοποιείται απευθείας ως index στον πίνακα των UUIDs.

Η εφαρμογή μας χρησιμοποιεί custom 128-bit UUIDs με μία κοινή βάση. Άρα ο πίνακας με τα UUIDs που παρέχεται από αυτή, περιέχει μόνο ένα UUID.

```
const struct att_uuid_128 wthrs_uuid [] =
{
    // WTHRS common UUID (LSB first)
    [WTHRS_UUID_IDX] = {{0x22, 0xef, 0xf5, 0x15, 0x22, 0x00, 0x5f, 0xb7,
                         0xe3, 0x11, 0x92, 0xf2, 0x00, 0x00, 0x98, 0xdc}},
};
```

Ακολουθεί ο ορισμός του master service και του temperature sensor service με τη βοήθεια των παραπάνω δομών. Τα υπόλοιπα ορίζονται με παρόμοιο τρόπο.

```
// Weather station service definition
const struct attm_ext_svc_def
wthrs_svc =
{
    ATTM_EXT_PRIMARY_SERVICE,

    .perm       = PERM(SVC, DISABLE),
    .uuid_type  = ATTM_EXT_UUID_IDX(WTHRS_UUID_IDX),
    .uuid       = WTHRS_UUID_WTHRS,

    .nb_inc_svc_128 = WTHRS_NB_SVC_INC,

    .nb_char    = 3,
    .svc_char   = wthrs_svc_char,
};



// Weather station service characteristics
const struct attm_ext_char_def
wthrs_svc_char [] =
{
    // Measurement
    {
    .prop       = ATT_CHAR_PROP_NTF,
    .uuid_type  = ATTM_EXT_UUID_IDX(WTHRS_UUID_IDX),
    .uuid       = WTHRS_UUID_MEASUREMENT,
    .max_length = WTHRS_MSRM_SIZE,
    .perm       = PERM(NTF, ENABLE),
    .desc_flags = ATTM_EXT_DESC_CLIENT_CFG,
    },

    // Measurement Interval
    {
    .prop       = ATT_CHAR_PROP_RD | ATT_CHAR_PROP_WR,
    .uuid_type  = ATTM_EXT_UUID_IDX(WTHRS_UUID_IDX),
    .uuid       = WTHRS_UUID_INTERVAL,
    .max_length = WTHRS_MSRM_INT_SIZE,
    .perm       = PERM(RD, ENABLE) | PERM(WR, ENABLE),
    },

    // Control Point
    {
    .prop       = ATT_CHAR_PROP_WR | ATT_CHAR_PROP_WR_NO_RESP
                  | ATT_CHAR_PROP_IND,
    .uuid_type  = ATTM_EXT_UUID_IDX(WTHRS_UUID_IDX),
    .uuid       = WTHRS_UUID_CTRL_PT,
    .max_length = WTHRS_CTRL_PT_MAXLEN,
    .perm       = PERM(WR, ENABLE) | PERM(IND, ENABLE),
    .desc_flags = ATTM_EXT_DESC_CLIENT_CFG,
    },

};



// Temperature sensor presentation format
const union attm_ext_desc_def
temp_snsr_prs_fmt =
{
    .prs_fmt = { WTHRS_TEMP_MSRM_TYPE, 0, WTHRS_TEMP_MSRM_UNIT }
};
```

```
// Temperature sensor service definition
const struct attm_ext_svc_def
temp_snsr_svc =
{
    ATTM_EXT_PRIMARY_SERVICE,

    .perm       = PERM(SVC, DISABLE),
    .uuid_type  = ATTM_EXT_UUID_IDX(WTHRS_UUID_IDX),
    .uuid       = WTHRS_UUID_TEMP_SNSR,

    .nb_char    = 3,
    .svc_char   = temp_snsr_svc_char,
};

// Temperature sensor service characteristics
const struct attm_ext_char_def
temp_snsr_svc_char [] =
{
    // Measurement
    {
    .prop       = ATT_CHAR_PROP_RD | ATT_CHAR_PROP_NTF,
    .uuid_type  = ATTM_EXT_UUID_IDX(WTHRS_UUID_IDX),
    .uuid       = WTHRS_UUID_TEMP_MEASUREMENT,
    .max_length = WTHRS_TEMP_MSRM_SIZE,
    .perm       = PERM(RD, ENABLE) | PERM(NTF, ENABLE),
    .desc_flags = ATTM_EXT_DESC_CLIENT_CFG | ATTM_EXT_DESC_PRS_FMT
                  | ATTM_EXT_DESC_PRS_FMT_DEF_NMSPC,
    .char_desc  = &temp_snsr_prs_fmt,
    },

    // Measurement Interval
    {
    .prop       = ATT_CHAR_PROP_RD | ATT_CHAR_PROP_WR,
    .uuid_type  = ATTM_EXT_UUID_IDX(WTHRS_UUID_IDX),
    .uuid       = WTHRS_UUID_TEMP_INTERVAL,
    .max_length = WTHRS_MSRM_INT_SIZE,
    .perm       = PERM(RD, ENABLE) | PERM(WR, ENABLE),
    },

    // Control Point
    {
    .prop       = ATT_CHAR_PROP_WR | ATT_CHAR_PROP_WR_NO_RESP
                  | ATT_CHAR_PROP_IND,
    .uuid_type  = ATTM_EXT_UUID_IDX(WTHRS_UUID_IDX),
    .uuid       = WTHRS_UUID_TEMP_CTRL_PT,
    .max_length = WTHRS_SNSR_CTRL_PT_MAXLEN,
    .perm       = PERM(WR, ENABLE) | PERM(IND, ENABLE),
    .desc_flags = ATTM_EXT_DESC_CLIENT_CFG,
    },

};
```

Στη διπλανή σελίδα, παρουσιάζεται μέρος της βάσης δεδομένων του Attribute Protocol, όπως προέκυψε μετά από service discovery, που πραγματοποιήθηκε με το εργαλείο ConnectionManager, το οποίο παρέχεται με το SDK του DA14580. Στην αρχή του ATT database, βλέπουμε διαδοχικά τα GAP service, GATT service, Device Information Service και Battery Service. Αυτά ακολουθούνται από το weather station (master) service, το οποίο με τη σειρά του ακολουθείται από τα sensor services. Στην εικόνα αυτή, μπορούμε να δούμε τη δομή του ATT database, τα UUIDs των attributes, καθώς και τα properties των χαρακτηριστικών, όπως αυτά ορίζονται στο αντίστοιχο characteristic declaration attribute.

0x0001	2800	-	Primary Service Definition
0x0002	2803	-	Characteristic Declaration
0x0003	2A00	Read	Device Name Characteristic
0x0004	2803	-	Characteristic Declaration
0x0005	2A01	Read	Appearance Characteristic
0x0006	2803	-	Characteristic Declaration
0x0007	2A02	Read, Write	Privacy Flags Characteristic
0x0008	2803	-	Characteristic Declaration
0x0009	2A04	Read	Preferred Conn. Params. Characteristic
0x000C	2800	-	Primary Service Definition
0x000D	2803	-	Characteristic Declaration
0x000E	2A05	Read, Indicate	Service Changed Characteristic
0x000F	2902	-	Client Characteristic Configuration
0x0010	2800	-	Primary Service Definition
0x0011	2803	-	Characteristic Declaration
0x0012	2A29	Read	Unknown UUID
0x0013	2803	-	Characteristic Declaration
0x0014	2A24	Read	Unknown UUID
0x0015	2803	-	Characteristic Declaration
0x0016	2A28	Read	Unknown UUID
0x0017	2803	-	Characteristic Declaration
0x0018	2A23	Read	Unknown UUID
0x0019	2800	-	Primary Service Definition
0x001A	2803	-	Characteristic Declaration
0x001B	2A19	Read, Notify	Unknown UUID
0x001C	2902	-	Client Characteristic Configuration
0x001D	2800	-	Primary Service Definition
0x001E	2802	-	Include Service Definition
0x001F	2802	-	Include Service Definition
0x0020	2802	-	Include Service Definition
0x0021	2803	-	Characteristic Declaration
0x0022	DC981001F29211E3B75F002215F5EF22	Notify	User-defined UUID
0x0023	2902	-	Client Characteristic Configuration
0x0024	2803	-	Characteristic Declaration
0x0025	DC981002F29211E3B75F002215F5EF22	Read, Write	User-defined UUID
0x0026	2803	-	Characteristic Declaration
0x0027	DC981003F29211E3B75F002215F5EF22	Write w/o Resp., Write, Indicate	User-defined UUID
0x0028	2902	-	Client Characteristic Configuration
0x0029	2800	-	Primary Service Definition
0x002A	2803	-	Characteristic Declaration
0x002B	DC981101F29211E3B75F002215F5EF22	Read, Notify	User-defined UUID
0x002C	2902	-	Client Characteristic Configuration
0x002D	2904	-	Characteristic Format
0x002E	2803	-	Characteristic Declaration
0x002F	DC981102F29211E3B75F002215F5EF22	Read, Write	User-defined UUID
0x0030	2803	-	Characteristic Declaration
0x0031	DC981103F29211E3B75F002215F5EF22	Write w/o Resp., Write, Indicate	User-defined UUID
0x0032	2902	-	Client Characteristic Configuration
0x0033	2800	-	Primary Service Definition
0x0034	2803	-	Characteristic Declaration
0x0035	DC981201F29211E3B75F002215F5EF22	Read, Notify	User-defined UUID
0x0036	2902	-	Client Characteristic Configuration
0x0037	2904	-	Characteristic Format
0x0038	2803	-	Characteristic Declaration
0x0039	DC981202F29211E3B75F002215F5EF22	Read, Write	User-defined UUID
0x003A	2803	-	Characteristic Declaration
0x003B	DC981203F29211E3B75F002215F5EF22	Write w/o Resp., Write, Indicate	User-defined UUID
0x003C	2902	-	Client Characteristic Configuration
0x003D	2800	-	Primary Service Definition
0x003E	2803	-	Characteristic Declaration
0x003F	DC981301F29211E3B75F002215F5EF22	Read, Notify	User-defined UUID
0x0040	2902	-	Client Characteristic Configuration
0x0041	2904	-	Characteristic Format
0x0042	2803	-	Characteristic Declaration
0x0043	DC981302F29211E3B75F002215F5EF22	Read, Write	User-defined UUID
0x0044	2803	-	Characteristic Declaration
0x0045	DC981303F29211E3B75F002215F5EF22	Write w/o Resp., Write, Indicate	User-defined UUID
0x0046	2902	-	Client Characteristic Configuration

Δομή της βάσης δεδομένων του Attribute Protocol.

7.4.2 Προφίλ

Προκειμένου να συνδέσουμε τις υπηρεσίες και τους αντίστοιχους αισθητήρες με τις τιμές που προβλέπονται από το προφίλ και, γενικότερα, για να διατηρήσουμε τον κώδικα υλοποίησης όσο το δυνατόν ανεξάρτητο των συγκεκριμένων αισθητήρων που περιλαμβάνονται στη συσκευή, ορίσαμε μια σειρά από δομές. Αυτές περιλαμβάνουν δεδομένα όπως η διεύθυνση αποθήκευσης και το μέγεθος της μέτρησης, η σημαία του αισθητήρα στο χαρακτηριστικό weather station measurement, και το αναγνωριστικό του στις εντολές του weather station control point. Επίσης παρέχονται τα handles των χαρακτηριστικών των services (ως offsets σε σχέση με το service declaration handle) και γίνεται αντιστοίχηση αυτών με τις λειτουργίες που προβλέπει το προφίλ κατά την εγγραφή των αντίστοιχων χαρακτηριστικών.

Για παράδειγμα, κατά την αποστολή μετρήσεων μέσω του master service measurement, επιλέγεται η κατάλληλη σημαία για κάθε αισθητήρα και το σωστό μέγεθος μέτρησης, χωρίς να χρειάζεται να γίνεται έλεγχος για τον τύπο του αισθητήρα. Ένα παράδειγμα μέσω του οποίου γίνεται φανερή η ανεξαρτησία του κώδικα από το είδος των αισθητήρων, είναι ο κώδικας που γράφει τα αποτελέσματα των μετρήσεων στο ATT database.

```
static void wthrs_set_msrm (const struct wthrs_msrm* msrm)
{
    wthrs_env.msrm = *msrm;

    for (int i = 0; i < WTHRS_NB_SNSR; ++i) {
        const struct wthrs_svc_data *s = &wthrs_svc_data[i];
        if (msrm->valid & snsr_bit_id(i)) {
            attmdb_att_set_value(wthrs_env.msrm_vhdl[i],
                                 s->msrm_size, s->msrm);
        }
    }
}
```

7.5 Wthrs Profile – Wthrs App

Τα wthrs profile και wthrs app αποτελούν τα δύο τμήματα του κώδικα υλοποίησης του weather station profile. Το wthrs profile είναι το ανεξάρτητο της εφαρμογής τμήμα, ενώ το wthrs app είναι το εξαρτώμενο από την εφαρμογή. Το wthrs profile υλοποεί τη γενική συμπεριφορά και τις διαδικασίες που προβλέπονται από το προφίλ και το BLE πρότυπο. Δημιουργεί τα sensor services και το master service στο ATT database, κατά την αρχικοποίηση της εφαρμογής, αναλαμβάνει την αποστολή μετρήσεων και αποτελεσμάτων εντολών στον collector, και ελέγχει τις εγγραφές στις τιμές των χαρακτηριστικών των υπηρεσιών, ενεργοποιώντας τις διαδικασίες που προβλέπονται στο weather station profile για κάθε περίπτωση. Το wthrs app αποτελεί το interface μεταξύ της εφαρμογής, του wthrs profile και του sensor manager, μεταφέροντας μηνύματα και εκκινώντας διαδικασίες σε αυτά. Υλοποιεί τη συμπεριφορά της εφαρμογής, που ορίζεται για τη συγκεκριμένη περίπτωση χρήσης. Οι βασικές του λειτουργίες είναι η μεταφορά των αποτελεσμάτων των μετρήσεων από τον sensor manager στο wthrs profile, ώστε να σταλούν στον collector, καθώς και η ενημέρωση του sensor manager, για αλλαγές στις παραμέτρους λειτουργίας του συστήματος, που τον επηρεάζουν. Επίσης, αναλαμβάνει την εκτέλεση των εντολών που ορίζονται στο προφίλ, και τις οποίες μπορεί να στείλει ο collector, γράφοντας κατάλληλες τιμές στο control point ενός service.

Το wthrs profile ορίζει ένα task και μια σειρά από μηνύματα και αντίστοιχα δεδομένα για την επικοινωνία του με το wthrs app. Το wthrs app χρησιμοποιεί το task της εφαρμογής. Τα βασικότερα μηνύματα και οι αντίστοιχες λειτουργίες είναι τα εξής:

- **WTHRS_CREATE_DB_REQ**: Στέλνεται από το wthrs app στο wthrs profile, κατά την αρχικοποίηση της εφαρμογής, προκειμένου να δημιουργηθούν οι υπηρεσίες του προφίλ που υποστηρίζονται από τη συσκευή. Το wthrs profile δημιουργεί τις υπηρεσίες στο ATT database και απαντά με το μήνυμα WTHRS_CREATE_DB_CFM.

- **WTHRS_CREATE_DB_CFM**: Στέλνεται από το wthrs profile στο wthrs app, με την ολοκλήρωση της δημιουργίας των υπηρεσιών. Το wthrs app ενημερώνει την εφαρμογή μέσω ενός μηνύματος APP_MODULE_INIT_CMP_EVT.

- **WTHRS_ENABLE_REQ**: Στέλνεται από το wthrs app στο wthrs profile, κατά τη σύνδεση με μια απομακρυσμένη συσκευή. Το wthrs profile ενεργοποιεί τις υπηρεσίες και θέτει κατάλληλες τιμές στις ρυθμίσεις του συστήματος που περιέχονται στο ATT database, όπως τα measurement intervals και οι τιμές των client characteristic configuration descriptors για το measurement και το control point κάθε υπηρεσίας.

- **GAPC_DISCONNECT_IND**: Στέλνεται από τον GAP controller της σύνδεσης στο wthrs profile, κατά την αποσύνδεση από την απομακρυσμένη συσκευή. Το wthrs profile απενεργοποιεί τις υπηρεσίες και ενημερώνει το wthrs app μέσω ενός μηνύματος WTHRS_DISABLE_IND.

- **WTHRS_MSRM_UPD_REQ**: Στέλνεται από το wthrs app στο wthrs profile, όταν υπάρχουν νέα αποτελέσματα μετρήσεων προς αποστολή. Το wthrs profile γράφει τις μετρήσεις στο ATT database και, στη συνέχεια, τις στέλνει στον collector, πραγματοποιώντας ένα ή περισσότερα GATT notifications, στα measurements των υπηρεσιών που προκύπτουν από τις τρέχουσες ρυθμίσεις του συστήματος, και με τον τρόπο που ορίζεται στο προφίλ. Αν οι μετρήσεις στέλνονται μέσω του master service measurement, τότε εκτελεί τη διαδικασία συγχώνευσης αυτών των μετρήσεων σε ένα πακέτο. Για παράδειγμα, αν υπάρχει μία μέτρηση θερμοκρασίας, ελέγχεται πρώτα αν είναι ενεργοποιημένα τα notifications στο temperature sensor measurement. Αν είναι ενεργοποιημένα, τότε η μέτρηση στέλνεται με notification του temperature measurement. Διαφορετικά, ελέγχεται αν είναι ενεργοποιημένα τα notifications στο master service measurement και αν η μέτρηση θερμοκρασίας μπορεί να σταλεί μέσω του master service. Σε αυτήν την περίπτωση, η μέτρηση στέλνεται, πιθανόν μαζί με άλλες μετρήσεις, με notification του master service measurement.

- **WTHRS_MSRM_INT_IND**: Στέλνεται από το wthrs profile στο wthrs app, όταν ο χρήστης μεταβάλει κάποιο από τα measurement intervals των υπηρεσιών. Το wthrs app ενημερώνει τον sensor manager για την αλλαγή της ρύθμισης.

- **WTHRS_NTF_CFG_IND**: Στέλνεται από το wthrs profile στο wthrs app, όταν ο χρήστης μεταβάλει την τιμή του client characteristic configuration στο measurement μιας υπηρεσίας, ενεργοποιώντας ή απενεργοποιώντας τα notifications και τις αντίστοιχες αποστολές. Τα wthrs profile και wthrs app ενημερώνουν τις τοπικές ρυθμίσεις τους.

- **WTHRS_IND_CFG_IND**: Στέλνεται από το wthrs profile στο wthrs app, όταν ο collector μεταβάλει την τιμή του client characteristic configuration στο control point μιας υπηρεσίας, ενεργοποιώντας ή απενεργοποιώντας τα indications. Τα wthrs profile και wthrs app ενημερώνουν τις τοπικές ρυθμίσεις τους.

- **WTHRS_CMD_IND**: Στέλνεται από το wthrs profile στο wthrs app, όταν ο collector στείλει κάποια εντολή, γράφοντας κατάλληλη τιμή στο control point μιας υπηρεσίας. Το wthrs app ελέγχει την εντολή και, αν αυτή δεν περιέχει κάποιο σφάλμα, εκκινεί την αντίστοιχη διαδικασία. Στην περίπτωση της εντολής "Update Now", το wthrs app στέλνει μήνυμα στον sensor manager για την εκκίνηση των απαραίτητων μετρήσεων. Δε στέλνεται απάντηση στο wthrs profile, ούτε indication στον collector. Στις υπόλοιπες εντολές, εκτελεί τη λειτουργία που προβλέπεται, και στέλνει το αποτέλεσμα στο wthrs profile με ένα μήνυμα τύπου WTHRS_CMD_RSP.

- **WTHRS_CMD_RSP**: Στέλνεται από το wthrs app στο wthrs profile, με την ολοκλήρωση μίας εντολής εκτός της "Update Now". Το wthrs profile στέλνει στον collector κατάλληλο indication, μέσω του control point της υπηρεσίας στην οποία δόθηκε η εντολή. Ταυτόχρονα, μηδενίζει τη σημαία εκκρεμούς εντολής, την οποία χρησιμοποιεί για την απαγόρευση έναρξης νέας εντολής πριν ολοκληρωθεί η προηγούμενη, όπως ορίζεται στο προφίλ.

- **GATTC_WRITE_CMD_IND**: Στέλνεται από τον GATT controller της σύνδεσης στο wthrs profile, κατά την εγγραφή, από τον collector, της τιμής ενός από τα χαρακτηριστικά των υπηρεσιών του προφίλ. Το wthrs profile ελέγχει ποιο χαρακτηριστικό είναι αυτό που μεταβάλλεται και ενημερώνει το wthrs app με ένα από τα παραπάνω μηνύματα. Πριν από αυτό όμως, πραγματοποιεί έλεγχο των τιμών που έστειλε ο collector. Αν υπάρχει κάποιο πρόβλημα, στέλνεται κατάλληλο μήνυμα λάθους και δεν ενημερώνεται η εφαρμογή. Στην περίπτωση των εντολών, ελέγχεται επίσης αν τα indications είναι ενεργοποιημένα στο αντίστοιχο control point και αν υπάρχει ήδη διαδικασία σε εξέλιξη. Και στις δύο περιπτώσεις στέλνεται κατάλληλο μήνυμα λάθους. Αν δεν υπάρχει σφάλμα και δεν πρόκειται για control point χαρακτηριστικό, τα δεδομένα που έστειλε ο collector γράφονται στο ATT database. Εκτός της περίπτωσης της εντολής "Update Now", η οποία μπορεί να σταλεί μέσω GATT Write Without Response, στις υπόλοιπες περιπτώσεις, στέλνεται στον collector ένα GATT Write Response.

7.6 Sensor Manager

Βασιζόμενοι στον παρόμοιο τρόπο λειτουργίας των διάφορων αισθητήρων για διαφορετικά μεγέθη, αποφασίσαμε να ορίσουμε ένα module, το οποίο ονομάσαμε sensor manager. Ο sensor manager αναλαμβάνει τον κεντρικό χειρισμό των αισθητήρων και των λειτουργιών τους, απλοποιώντας σε μεγάλο βαθμό τον κώδικα που απαιτείται για κάθε συγκεκριμένο αισθητήρα.

7.6.1 Αρχές λειτουργίας

Οι βασικές λειτουργίες ενός αισθητήρα είναι η αρχικοποίησή του, η έναρξη και ο τερματισμός της μέτρησης και η λήψη των αποτελεσμάτων. Οι λειτουργίες αυτές γίνονται με διαφορετικό τρόπο σε κάθε αισθητήρα, όπως ορίζεται στο specification του, το οποίο παρέχεται από την εταιρία κατασκευής του. Επίσης, το προφίλ δίνει τη δυνατότητα ενεργοποίησης, απενεργοποίησης και επαναφοράς αισθητήρων, αν αυτό υποστηρίζεται από τη συσκευή. Ο sensor manager παρέχει κατάλληλες διαδικασίες για την υλοποίησή τους.

Κατά τη λειτουργία ενός αισθητήρα μπορεί να υπεισέρχονται διάφορες χρονικές καθυστερήσεις. Για παράδειγμα, ένας αισθητήρας μπορεί να απαιτεί κάποιο χρονικό διάστημα, από τη στιγμή της ενεργοποίησής του, μέχρι την αποστολή της πρώτης εντολής

προς αυτόν. Επίσης, για κάθε αισθητήρα, ορίζεται ο χρόνος που μεσολαβεί από την έναρξη της μέτρησης, μέχρι τα αποτελέσματα αυτής να είναι διαθέσιμα. Ο sensor manager αναλαμβάνει τον έλεγχο όλων αυτών των χρονικών καθυστερήσεων.

Ο sensor manager ορίζει επίσης μια σειρά από timers, που παρέχονται από τον kernel του DA14580. Υπάρχει ένας timer για κάθε sensor service και ένας κεντρικός timer για τις μετρήσεις που ελέγχονται από το master service. Οι timers αυτοί χρησιμοποιούνται για την έναρξη των περιοδικών μετρήσεων, με περιόδους που ορίζονται από τον χρήστη, μέσω των measurement intervals των υπηρεσιών, και με βάση τους κανόνες που προβλέπονται στο προφίλ. Δηλαδή, μόνο μετρήσεις αισθητήρων που ελέγχονται από το master service μπορούν να ξεκινήσουν κατά την περιοδική μέτρηση του τελευταίου. Σε κάθε περίπτωση, αν ένας αισθητήρας έχει ενεργοποιημένες τις περιοδικές μετρήσεις (μη μηδενικό measurement interval), αυτές εκκινούνται πάντοτε ξεχωριστά από τις υπόλοιπες. Αυτό είναι ανεξάρτητο του τρόπου αποστολής των αποτελεσμάτων, αν δηλαδή το αποτέλεσμα της μέτρησης στέλνεται μέσω του sensor ή του master service measurement. Οι timers των περιοδικών μετρήσεων ενεργοποιούνται κατά τη σύνδεση με τον collector και απενεργοποιούνται κατά την αποσύνδεση.

Ο sensor manager αναλαμβάνει το συγχρονισμό της λειτουργίας των αισθητήρων, καλώντας, όπου είναι απαραίτητο, συναρτήσεις που ορίζονται για κάθε αισθητήρα και υλοποιούν τη διακριτή λειτουργία του. Επιπλέον, παρέχει δυνατότητες, όπως αναβολή μέτρησης, αν υπάρχει ήδη μέτρηση σε εξέλιξη, και αναμονή για την ολοκλήρωση διαφορετικών μετρήσεων που συμβαίνουν ταυτόχρονα, ώστε να σταλούν όλες μαζί.

7.6.2 Καταστάσεις αισθητήρων

Ένας αισθητήρας, υπό τον έλεγχο του sensor manager, βρίσκεται κάθε στιγμή σε μία από τις καταστάσεις που ορίζονται σε αυτόν. Οι καταστάσεις είναι οι εξής:

- **SNSR_STANDBY**: Αρχική κατάσταση. Ο αισθητήρας είναι απενεργοποιημένος.
- **SNSR_START**: Κατάσταση ενεργοποίησης . Καλείται η συνάρτηση ενεργοποίησης του αισθητήρα. Μετα την ολοκλήρωση της ενεργοποίησης ο αισθητήρας περνάει σε κατάσταση SNSR_INIT.
- **SNSR_STOP**: Κατάσταση απενεργοποίησης. Καλείται η συνάρτηση απενεργοποίησης του αισθητήρα. Μετα την ολοκλήρωση της απενεργοποίησης ο αισθητήρας περνάει σε κατάσταση SNSR_STANDBY.
- **SNSR_INIT**: Κατάσταση αρχικοποίησης. Καλείται η συνάρτηση αρχικοποίησης του αισθητήρα.
- **SNSR_IDLE**: Κατάσταση αναμονής. Ο αισθητήρας είναι έτοιμος να εκτελέσει μέτρηση. Είναι η μόνη κατάσταση στην οποία μπορούν να εκτελεστούν μετρήσεις.
- **SNSR_MEASURE**: Κατάσταση μέτρησης.
- **SNSR_RECOVERY**: Κατάσταση recovery μετά από μέτρηση. Για κάποιο χρονικό διάστημα μετά την ολοκλήρωση μιας μέτρησης, το οποίο ορίζεται από τον αισθητήρα, δεν επιτρέπεται έναρξη νέας μέτρησης σε αυτόν. Σε αυτήν την περίπτωση, ο sensor manager χρησιμοποιεί το αποτέλεσμα της τελευταίας μέτρησης.

7.6.3 Event Dispatcher

Πέρα από τους περιοδικούς timers που ορίζει, ο sensor manager πρέπει, επίσης, να ελέγχει και τις χρονικές καθυστερήσεις που προκύπτουν στη λειτουργία των αισθητήρων, όπως ο χρόνος εκκίνησης και ο χρόνος μέτρησης. Προκειμένου να πραγματοποιηθεί η λειτουργικότητα αυτή, αλλά και για την υποστήριξη γενικότερων λειτουργιών του sensor

manager, υλοποιήσαμε έναν απλό event dispatcher. Με τη βοήθεια του event dispatcher μπορούν να οριστούν events με συγκεκριμένα timeouts, τα οποία τοποθετούνται σε μια ουρά και εκτελούνται όταν εκπνεύσει το timeout. Για τη λειτουργία του event dispatcher, χρησιμοποιείται ένας επιπλέον timer. Έτσι, για παράδειγμα, κατά την έναρξη μιας μέτρησης ο sensor manager τοποθετεί στην ουρά του event dispatcher ένα SNSR_EVT_MSRM_STOP event, το οποίο δηλώνει το τέλος της μέτρησης. Το timeout τίθεται στο χρόνο μέτρησης του αισθητήρα. Με τη λήξη του, ο dispatcher καλεί την αντίστοιχη συνάρτηση του manager. Ο manager, αφού τερματίσει τη μέτρηση και λάβει το αποτέλεσμα, θέτει τον αισθητήρα σε κατάσταση SNSR_RECOVERY, και τοποθετεί στην ουρά του dispatcher ένα SNSR_EVT_CHG_STATE event, με το οποίο, μετά από το ορισμένο από τον αισθητήρα timeout, η κατάστασή του επανέρχεται σε SNSR_IDLE. Ο event dispatcher μπορεί να χρησιμοποιηθεί και για την εκτέλεση λειτουργιών από τον manager. Για παράδειγμα, δίνεται η δυνατότητα στους αισθητήρες να ορίζουν callback συναρτήσεις που καλούνται μετά από το event timeout.

7.6.4 Ορισμός αισθητήρων

Κάθε αισθητήρας που ελέγχεται από τον sensor manager, ορίζεται μέσω μιας δομής, που περιλαμβάνει δεδομένα και συναρτήσεις, με τη βοήθεια των οποίων μπορεί ο sensor manager να ελέγξει όλες τις λειτουργίες του αισθητήρα. Στα δεδομένα περιλαμβάνονται τιμές χρονικής καθυστέρησης, ενώ οι συναρτήσεις υλοποιούν τη διακριτή λειτουργία του κάθε αισθητήρα. Οι τελευταίες, επειδή δε χρειάζεται να εκτελούν οι ίδιες τις επιπλέον λειτουργίες που αναλαμβάνει ο manager, μπορεί να είναι πολύ απλές (π.χ. γράψιμο ή διάβασμα τιμών από το I^2C bus). Ακολουθεί ένα παράδειγμα ορισμού αισθητήρα:

```
const struct sensor_data wthrs_snsr_data [WTHRS_NB_SNSR] =
{
...
// ==================== Humidity ====================
    [WTHRS_HUMID_SNSR] =
    {
    .snsr = WTHRS_HUMID_SNSR,
    .boot_time  = WTHRS_SNSR_DELAY,
    .stop_time  = WTHRS_SNSR_DELAY,
    .msrm_time  = WTHRS_HUMID_MSRM_TIME,
    .rcvr_time  = WTHRS_SNSR_RECOVERY_TIME,
    .concurrent = SNSMGR_EXCLUSIVE_MSRM,
    .msrm       = app_wthrs_env.msrm.humidity,
    .msrm_fmt   = WTHRS_HUMID_MSRM_TYPE,
    .start      = wthrs_snsr_humid_start,
    .stop       = wthrs_snsr_humid_stop,
    .init       = wthrs_snsr_humid_init,
    .msrm_start = wthrs_snsr_humid_msrm,
    .msrm_stop  = wthrs_snsr_humid_msrm_stop,
    .msrm_get   = wthrs_snsr_humid_msrm_get,
    .callback   = NULL,
    },
...
};
```

Μια συνάρτηση που δεν υλοποιείται μπορεί να τίθεται σε NULL. Τα δεδομένα και οι βασικότερες συναρτήσεις, που περιέχονται στην παραπάνω δομή, είναι τα ακόλουθα:

- **boot_time**: Χρόνος ενεργοποίησης.
- **stop_time**: Χρόνος απενεργοποίησης.
- **msrm_time**: Διάρκεια μέτρησης.
- **rcvr_time**: Διάρκεια παραμονής σε κατάσταση SNSR_RECOVERY.
- **concurrent**: Bitfield που ορίζει αισθητήρες με τους οποίους επιτρέπεται ταυτόχρονη μέτρηση. Η τιμή SNSMGR_EXCLUSIVE_MSRM (0) σημαίνει ότι ο αισθητήρας δεν επιτρέπεται να μετρά ταυτόχρονα με άλλους αισθητήρες. Η αποτροπή ταυτόχρονων μετρήσεων δίνει τη δυνατότητα μείωσης του μέγιστου ρεύματος κατανάλωσης της συσκευής.
- **msrm**: Διεύθυνση αποθήκευσης των αποτελεσμάτων.
- **msrm_fmt**: Format της τιμής των αποτελεσμάτων που θα αποθηκευτούν στην παραπάνω διεύθυνση.
- **start**: Συνάρτηση ενεργοποίησης αισθητήρα, για την περίπτωση που η δυνατότητα αυτή υποστηρίζεται από τη συσκευή.
- **stop**: Συνάρτηση απενεργοποίησης αισθητήρα, για την περίπτωση που η δυνατότητα αυτή υποστηρίζεται από τη συσκευή.
- **init**: Συνάρτηση αρχικοποίησης αισθητήρα, όπως προβλέπεται από το specification του.
- **msrm_start**: Συνάρτηση έναρξης μέτρησης.
- **msrm_stop**: Συνάρτηση τερματισμού μέτρησης.
- **msrm_get**: Συνάρτηση λήψης του αποτελέσματος και αποθήκευσής του στη διεύθυνση msrm.

7.6.5 Sensor Manager Task

Τέλος, ο sensor manager ορίζει ένα task και ένα σύνολο μηνυμάτων και αντίστοιχων δεδομένων, με τα οποία επικοινωνεί με το wthrs app για τη λήψη εντολών και την αποστολή δεδομένων και αποτελεσμάτων. Τα βασικότερα μηνύματα είναι τα ακόλουθα:

- **SNSMGR_INIT_REQ**: Στέλνεται από το wthrs app στον sensor manager, κατά την αρχικοποίηση της εφαρμογής (app_init_func). Ο sensor manager αρχικοποιεί τοπικά δεδομένα και ρυθμίσεις, στις τιμές που παρέχονται από την εφαρμογή.

- **SNSMGR_START_REQ**: Στέλνεται από το wthrs app στον sensor manager, για την ενεργοποίηση των αισθητήρων που ορίζονται στα δεδομένα του μηνύματος. Το μήνυμα αυτό στέλνεται μετά από αντίστοιχη εντολή του collector, ή κατά την αρχικοποίηση του ATT database (app_db_init_func), προκειμένου οι αισθητήρες να είναι διαθέσιμοι με την εκκίνηση του advertising. Ο sensor manager ενεργοποιεί τους αισθητήρες που ζητούνται και απαντά με το μήνυμα SNSMGR_START_IND.

- **SNSMGR_START_IND**: Στέλνεται από τον sensor manager στο wthrs app, με την ολοκλήρωση της ενεργοποίησης των αισθητήρων. Αν η εφαρμογή βρίσκεται σε φάση αρχικοκοποίησης του ATT database, το wthrs app ενημερώνει την εφαρμογή μέσω ενός μηνύματος APP_MODULE_INIT_CMP_EVT.

- **SNSMGR_STOP_REQ**: Στέλνεται από το wthrs app στον sensor manager, μετά από εντολή του collector, για την απενεργοποίηση των αισθητήρων που ορίζονται στα δεδομένα του μηνύματος. Ο sensor manager απενεργοποιεί τους αισθητήρες που ζητούνται και απαντά με το μήνυμα SNSMGR_STOP_IND.

- **SNSMGR_RESTART_REQ**: Στέλνεται από το wthrs app στον sensor manager, μετά από εντολή του collector, για την επαναφορά των αισθητήρων που ορίζονται

στα δεδομένα του μηνύματος. Ο sensor manager επανεκκινεί τους αισθητήρες που ζητούνται και απαντά με το μήνυμα SNSMGR_START_IND.

- **SNSMGR_CONNECTION_IND**: Στέλνεται από το wthrs app στον sensor manager, κατά τη σύνδεση με μια απομακρυσμένη συσκευή. Ο sensor manager ενεργοποιεί τους timers των περιοδικών μετρήσεων.

- **GAPC_DISCONNECT_IND**: Στέλνεται από τον GAP controller της σύνδεσης στον sensor manager, κατά την αποσύνδεση από την απομακρυσμένη συσκευή. Ο sensor manager απενεργοποιεί τους timers των περιοδικών μετρήσεων.

- **SNSMGR_MSRM_UPD_REQ**: Στέλνεται από το wthrs app στον sensor manager, μετά από εντολή "Update Now" του collector, για την άμεση έναρξη μέτρησης σε έναν ή περισσότερους αισθητήρες, που ορίζονται στα δεδομένα του μηνύματος. Ο sensor manager εκκινεί τις μετρήσεις, λαμβάνοντας υπόψη την τρέχουσα κατάσταση των αισθητήρων. Αν για παράδειγμα ένας αισθητήρας είναι σε κατάσταση recovery, δεν εκκινείται νέα μέτρηση, αλλά απλώς στέλνεται το αποτέλεσμα της τελευταίας.

- **SNSMGR_MSRM_UPD_IND**: Στέλνεται από τον sensor manager στο wthrs app, όταν υπάρχουν νέα δεδομένα μετρήσεων, είτε περιοδικών, είτε μετά από εντολή "Update Now". Το wthrs app στέλνει τα αποτελέσματα στο wthrs profile, ώστε να σταλούν μέσω notification στον collector.

- **SNSMGR_MSRM_INT_IND**: Στέλνεται από το wthrs app στον sensor manager, όταν μεταβάλλεται το measurement interval ενός service. Ο sensor manager ενημερώνει τις τοπικές ρυθμίσεις και, αν είναι απαραίτητο, ενεργοποιεί ή απενεργοποιεί τους αντίστοιχους timers.

- **SNSMGR_MASTER_MSRM_IND**: Στέλνεται από το wthrs app στον sensor manager, όταν μεταβαλλεται η ρύθμιση των μετρήσεων που ελέγχονται από το master service. Ο sensor manager ενημερώνει τις τοπικές ρυθμίσεις και, αν είναι απαραίτητο, ενεργοποιεί ή απενεργοποιεί τον timer του master service.

- **SNSMGR_SYNC_SNSR_MSRM_REQ**: Στέλνεται από το wthrs app στον sensor manager, για τον συγχρονισμό των μετρήσεων, μετά από αντίστοιχη εντολή του collector.

- **SNSMGR_EVT_TIMER**: Στέλνεται από τον kernel στον sensor manager με τη λήξη του event timer. Ο sensor manager ελέγχει τα event timeouts και, αν είναι απαραίτητο, εκτελεί event dispatching.

- **SNSMGR_PRD_TIMER**: Αποτελεί το πρώτο από μία σειρά μηνυμάτων, που χρησιμοποιούνται για τον ορισμό των timers του master service και των αισθητήρων. Στέλνεται από τον kernel στον sensor manager με τη λήξη ενός από τους ενεργοποιημένους timer, οπότε ο sensor manager εκκινεί την περιοδική μέτρηση στον αντίστοιχο αισθητήρα.

7.7 Αισθητήρες

Σε αυτή την υπο-ενότητα, κάνουμε μια γενική περιγραφή των αισθητήρων, του τρόπου λειτουργίας τους και σύνδεσής τους με το DA14580 SoC, και του τρόπου επικοινωνίας τους με την εφαρμογή.

T5403 Temperature – Barometric Pressure Sensor

7.7.1 Κριτήρια επιλογής

Βασικό κριτήριο για την επιλογή των αισθητήρων υπήρξε η δυνατότητά τους να λειτουργούν με την μπαταρία του συστήματος, δηλαδή με τάση από 2.7V ως 3.0V, όση είναι και η τυπική τάση εξόδου μιας μπαταρίας τύπου coin-cell, όπως η CR2032. Η τάση εξόδου, στις μπαταρίες αυτού του τύπου, πέφτει καθώς πλησιάζουν προς το τέλος της διάρκειας ζωής τους. Εκτός από την τάση λειτουργίας, μεγάλο ρόλο έπαιξε και η συνολική και μέγιστη κατανάλωση του κάθε αισθητήρα, ώστε η παρουσία του να μην αναιρεί τα ενεργειακά οφέλη της χρησιμοποίησης ενός προτύπου όπως το BLE.

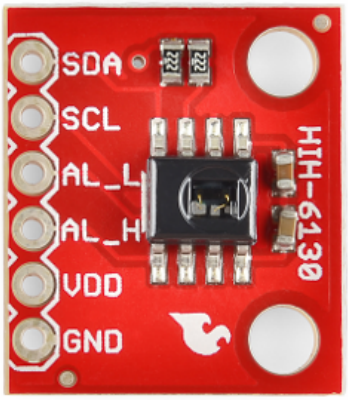
Για τον λόγο αυτό, προτιμήθηκαν κυρίως ψηφιακοί αισθητήρες, αφού υπάρχουν αρκετοί που ικανοποιούν τόσο το κριτήριο της τάσης λειτουργίας, όσο και της κατανάλωσης. Επίσης, οι ψηφιακοί αισθητήρες δίνουν τα αποτελέσματα των μετρήσεών τους, ίσως μετά από κάποια επεξεργασία που απαιτείται, απευθείας σε μονάδες του συγκεκριμένου μεγέθους το οποίο μετράνε. Αυτό οδηγεί σε μεγαλύτερη ακρίβεια αποτελεσμάτων. Υπάρχουν, βέβαια, και αναλογικοί αισθητήρες οι οποίοι δεν παρουσιάζουν υψηλή κατανάλωση και θα μπορούσαν να χρησιμοποιηθούν για τη μέτρηση μεγεθών, στα οποία η ακρίβεια των αποτελεσμάτων δεν είναι τόσο σημαντική. Τέτοια μεγέθη είναι για παράδειγμα η ταχύτητα και η διεύθυνση του ανέμου. Αν η ταχύτητα του ανέμου παρουσιάζεται στον χρήστη σε μποφόρ, τότε η ύπαρξη μεγαλύτερης ακρίβειας δε θα άλλαζε τα αποτελέσματα. Επίσης, στην περίπτωση της διεύθυνσης του ανέμου, δε χρειάζεται μεγάλη ακρίβεια, αφού αυτό που ενδιαφέρει συνήθως, είναι ο κατά προσέγγιση ορισμός της διεύθυνσης (π.χ. NE, SSW, S, SE κλπ.) και όχι ο ακριβής προσδιορισμός της. Αντίθετα, σε μεγέθη όπως η θερμοκρασία και η υγρασία, η ακρίβεια των μετρήσεων θεωρείται σημαντική.

Θα πρέπει, εδώ, να σημειωθεί ότι οι αισθητήρες, που χρησιμοποιήθηκαν για την υλοποίηση του συστήματος, έχουν τη μορφή που παρουσιάζεται στην παραπάνω εικόνα, δηλαδή κάθε αισθητήρας περιλαμβάνεται στη δική του πλακέτα. Σε κάποιες περιπτώσεις τα κυκλώματα αυτά, εκτός του chipset του αισθητήρα, μπορεί να περιλαμβάνουν και επιπλέον

στοιχεία, όπως voltage regulators και transistors σύνδεσης στον διάδρομο, προκειμένου να μπορούν να χρησιμοποιηθούν από micro-controllers με διαφορετική τάση εξόδου υψηλής κατάστασης. Τα επιπλέον αυτά στοιχεία καταναλώνουν ενέργεια, οπότε η συνολική κατανάλωση του συστήματος δεν είναι η μικρότερη δυνατή. Σε ένα πραγματικό σύστημα τέτοιου τύπου, οι αισθητήρες θα βρίσκονταν στην ίδια πλακέτα με το DA14580, δημιουργώντας μια μικρού μεγέθους συσκευή.

7.7.2 Τρόποι σύνδεσης

Οι ψηφιακοί αισθητήρες, που επιλέχθηκαν, συνδέονται με το DA14580 μέσω του I^2C bus και η επικοινωνία μαζί τους επιτυγχάνεται με το αντίστοιχο πρότυπο επικοινωνίας, που περιγράφεται συνοπτικά παρακάτω. Ένα πλεονέκτημα αυτού του τρόπου σύνδεσης είναι ότι επιτρέπει σε πολλούς αισθητήρες να χρησιμοποιούν τον ίδιο διάδρομο για την επικοινωνία τους με το σύστημα, χωρίς να δημιουργείται κάποιο πρόβλημα. Αυτό βοηθάει στη μείωση του αριθμού των GPIO pins, τα οποία πρέπει να χρησιμοποιηθούν για τη σύνδεση των αισθητήρων, και τα οποία, ειδικά στη μικρότερη σε μέγεθος εκδοχή του DA14580, είναι περιορισμένα. Επίσης το I^2C bus καταναλώνει ελάχιστη ενέργεια όταν δε χρησιμοποιείται. Για τη σύνδεση αναλογικών αισθητήρων μπορεί να χρησιμοποιηθεί το ADC περιφερειακό του DA14580 και ο αντίστοιχος driver που παρέχεται για αυτό. Το ADC κύκλωμα, το οποίο χρησιμοποιείται, επίσης, για τη μέτρηση του επιπέδου της μπαταρίας, μπορεί να μετρήσει τάσεις μέχρι 3.6V με κατάλληλη ρύθμιση. Η τάση αυτή μπορεί να είναι, είτε σε σχέση με τη γη, είτε μεταξύ δύο διαφορετικών pins.



HIH-6130 Humidity – Temperature Sensor

7.7.3 I^2C Bus

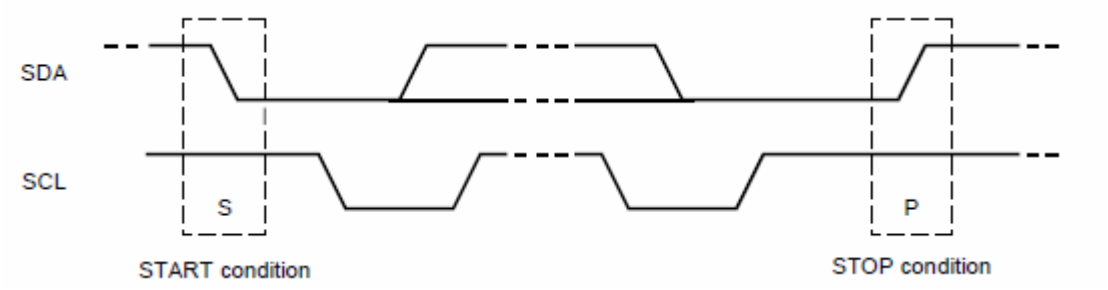
Το I^2C είναι ένα σχετικά απλό πρότυπο επικοινωνίας, πάνω από ένα διάδρομο που αποτελείται από δύο καλώδια, ένα για σήματα δεδομένων και ένα για σήματα ρολογιού. Τα σήματα αυτά ονομάζονται αντίστοιχα SDA και SCL. Το πρότυπο ορίζει τον τρόπο με τον οποίο πολλαπλές συσκευές μπορούν να συνδέονται στον διάδρομο και, χρησιμοποιώντας κατάλληλους συνδυασμούς των δύο παραπάνω σημάτων, να ανταλλάσσουν bytes δεδομένων.

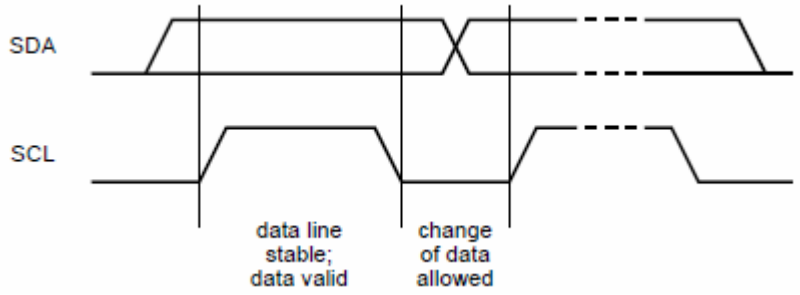
Οι συσκευές συνδέονται πάνω στον διάδρομο με ένα κύκλωμα εξόδου τύπου ανοιχτού συλλέκτη (open collector) ή ανοιχτής υποδοχής (open drain), που δημιουργεί ένα κύκλωμα wired-AND. Ο διάδρομος είναι συνδεμένος με την τάση εισόδου χρησιμοποιώντας αντιστάσεις pull-up. Αυτό σημαίνει ότι ο διάδρομος είναι συνεχώς σε υψηλή κατάσταση, εκτός αν κάποια συσκευή συνδέσει την έξοδό της με τη γη, οπότε περνάει σε χαμηλή

κατάσταση. Επειδή τα κυκλώματα τύπου open collector ή open drain δεν τραβάνε ρεύμα από τον διάδρομο, όταν τα αντίστοιχα transistors είναι κλειστά, το I^2C bus έχει πολύ χαμηλή κατανάλωση ενέργειας όταν δε χρησιμοποιείται. Όταν είναι σε χρήση, υπάρχει κατανάλωση μόνο όταν ο διάδρομος τίθεται χαμηλά, η οποία εξαρτάται από τις τιμές των αντιστάσεων pull-up που χρησιμοποιούνται.

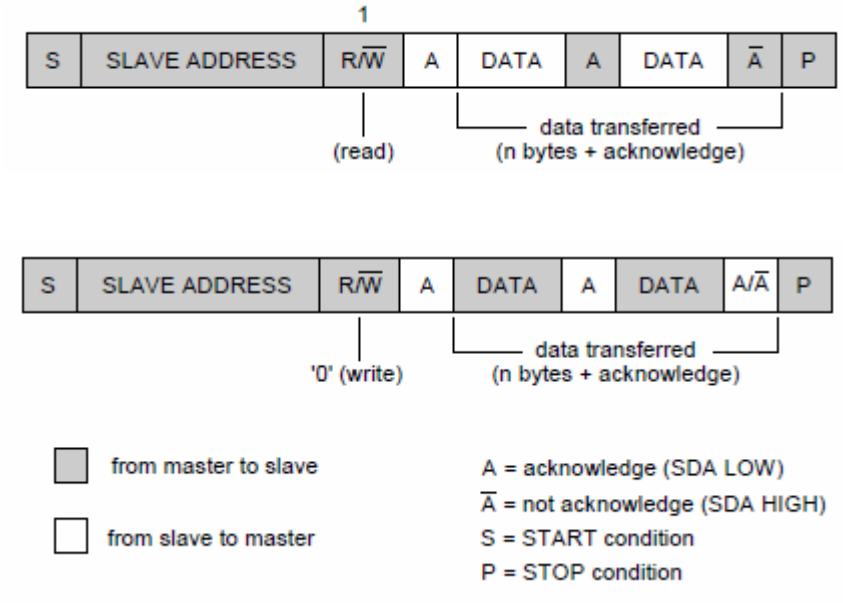
Ο διάδρομος ελέγχεται από μια συσκευή, που ονομάζεται master, και η οποία μπορεί να στέλνει ή να λαμβάνει δεδομένα από όλες τις υπόλοιπες συσκευές του διαδρόμου, που ονομάζονται slaves. Το πρότυπο δίνει τη δυνατότητα ύπαρξης πολλαπλών master σε έναν διάδρομο, όμως μόνο ένας μπορεί κάθε χρονική στιγμή να έχει τον έλεγχο του διαδρόμου. Το πρότυπο καθορίζει τον τρόπο, με τον οποίο επιλύονται τυχόν συγκρούσεις μεταξύ masters, που προσπαθούν να χρησιμοποιήσουν τον διάδρομο ταυτόχρονα. Οι συσκευές slave δεν μπορούν να επικοινωνήσουν μεταξύ τους, παρά μόνο να ακολουθήσουν τις οδηγίες του master, διαβάζοντας ή γράφοντας δεδομένα στον διάδρομο.

Η επικοινωνία ξεκινά με μια συγκεκριμένη ακολουθία σημάτων στον διάδρομο, η οποία ονομάζεται start condition, και τελειώνει με μια διαφορετική ακολουθία σημάτων, η οποία ονομάζεται stop condition. Μεταξύ των start και stop conditions, το SCL χρησιμοποιείται για να καταδείξει πότε το SDA ορίζει έγκυρα δεδομένα.





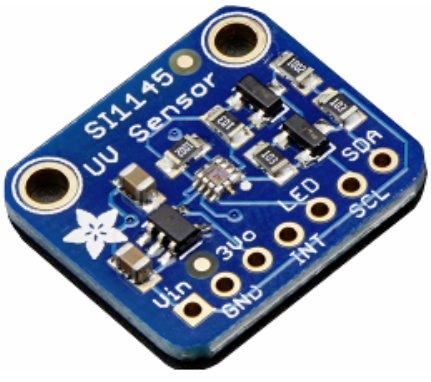
Κάθε slave έχει μία διεύθυνση των 7 ή 10 bit, την οποία ο master γράφει στον διάδρομο, ακολουθούμενη από ένα bit, που καθορίζει τη λειτουργία (read ή write), που θέλει να πραγματοποιήσει ο master. Ο slave διαβάζει τα δεδομένα από τον διάδρομο και, αν δει τη δική του διεύθυνση μετά από start condition, απαντά στον master (ACK), θέτοντας το SDA χαμηλά. Από εκεί και πέρα, η διαδικασία εξαρτάται από τη λειτουργία που επέλεξε ο master. Αν πρόκειται για write, ο master γράφει δεδομένα στον διάδρομο τα οποία λαμβάνει ο slave, ενώ, αν πρόκειται για read, ο slave είναι αυτός που γράφει δεδομένα στον διάδρομο προκειμένου να διαβαστούν από τον master. Τα δεδομένα στέλνονται ένα byte κάθε φορά, ακολουθούμενο από απάντηση (ACK ή NACK) της άλλης πλευράς. Το NACK τερματίζει την αποστολή δεδομένων.



Στις συνηθισμένες περιπτώσεις χρήσης, για μια συσκευή slave, ορίζεται στο specification της μια σειρά από διευθύνσεις (registers), τις οποίες μπορεί να χρησιμοποιήσει μια συσκευή master, προκειμένου να εκτελέσει λειτουργίες πάνω σε αυτήν και να λάβει αποτελέσματα. Αυτό πραγματοποιείται με την εγγραφή ή την ανάγνωση τιμών από τους registers που ορίζονται. Για την εγγραφή σε ένα register, ο master γράφει τη διεύθυνσή του στον διάδρομο, ακολουθουμένη από ένα ή περισσότερα bytes δεδομένων προς εγγραφή. Για την ανάγνωση από ένα register, ο master γράφει και πάλι τη διεύθυνσή του στον διάδρομο, αλλά στη συνέχεια εκκινεί μία λειτουργία read, κατά την οποία ο slave γράφει στον διάδρομο τα δεδομένα του register.

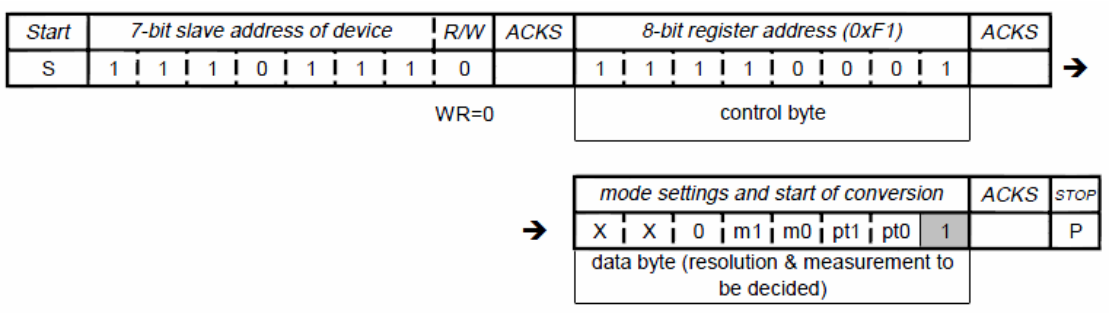
7.7.4 Sensor Drivers

Κάθε ψηφιακός αισθητήρας είναι, λοιπόν, μια συσκευή I^2C slave, με συγκεκριμένη διεύθυνση, η οποία είναι, συνήθως, hardcoded ή με μικρή δυνατότητα παραμετροποίησης μέσω pins. Η διεύθυνση πρέπει να είναι διαφορετική για κάθε αισθητήρα που συνδέεται στον διάδρομο. Στο specification του αισθητήρα, ορίζονται διευθύνσεις registers και τα δεδομένα που περιέχονται σε αυτούς ή οι λειτουργίες που εκτελούνται με την εγγραφή κατάλληλων τιμών. Έτσι, το DA14580, λειτουργώντας ως master του διαδρόμου, μπορεί, για παράδειγμα, γράφοντας κατάλληλη τιμή σε έναν register, να δώσει εντολή σε έναν αισθητήρα για εκκίνηση της μέτρησης, και, αφού περιμένει το απαραίτητο χρονικό διάστημα, να λάβει το αποτέλεσμα, διαβάζοντας τις τιμές συγκεκριμένων registers.

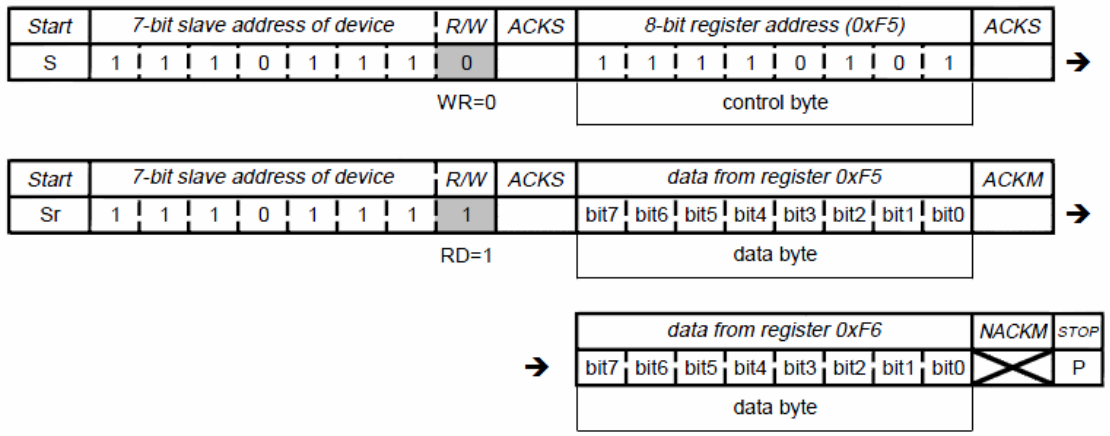


Si1145 Ambient Light – UV Index Sensor

Για παράδειγμα, στο T5403, το οποίο έχει διεύθυνση 0x77, ορίζεται η ακόλουθη επικοινωνία μέσω I^2C για την πραγματοποίηση μέτρησης και τη λήψη των αποτελεσμάτων.



Εντολή έναρξης μέτρησης



Λήψη αποτελεσμάτων

Βασιζόμενοι, λοιπόν, στο specification κάθε αισθητήρα, υλοποιήσαμε τον αντίστοιχο driver, και προσθέσαμε τις απαραίτητες συναρτήσεις και δεδομένα στον sensor manager.

Κεφάλαιο 8

Android App: Weather Collector

Για την υλοποίηση του δεύτερου τμήματος του συστήματος καταγραφής καιρού, το οποίο αποτελεί το user interface του συστήματος, χρησιμοποιήθηκε το λειτουργικό σύστημα Android, πάνω στο οποίο δημιουργήθηκε μια εφαρμογή (app), βασισμένη στο Bluetooth Low Energy API, που παρέχεται στις τελευταίες εκδόσεις της πλατφόρμας. Η εφαρμογή αυτή μπορεί να εκτελείται σε τελευταίας γενιάς Android smartphones ή tablets, με την προϋπόθεση να παρέχουν υποστήριξη για το BLE. Η εφαρμογή παρουσιάζει στον χρήστη του συστήματος τα αποτελέσματα των μετρήσεων, τα οποία αποστέλλονται από τη συσκευή καταγραφής, μέσω του πρωτοκόλλου BLE, με τον τρόπο και τη μορφή που ορίζεται από το Weather Station Profile, που παρουσιάσαμε σε προηγούμενο κεφάλαιο. Η παρουσίαση των αποτελεσμάτων στον χρήστη γίνεται μέσα από κατάλληλα σχεδιασμένο user interface, το οποίο παρέχει επιπλέον δυνατότητες, όπως η προβολή του ιστορικού μετρήσεων και η δημιουργία γραφικών παραστάσεων. Παράλληλα, η εφαρμογή υλοποιεί τις διαδικασίες και τη λειτουργικότητα που αναμένεται από αυτή, με βάση το ρόλο που έχει στο Weather Station Profile. Έτσι, δίνει στον χρήστη τη δυνατότητα παραμετροποίησης του συστήματος, παρέχοντας κατάλληλο interface για τον καθορισμό όλων των παραμέτρων που προβλέπονται στο προφίλ, τις οποίες μεταφέρει στη συσκευή καταγραφής, με τη βοήθεια εγγραφών στα απαιτούμενα από κάθε ρύθμιση χαρακτηριστικά.

Η εφαρμογή, χρησιμοποιώντας τη BLE στοίβα πρωτοκόλλων του Android και το Bluetooth hardware της συσκευής, μπορεί να εκτελέσει αναζήτηση (scanning) BLE συσκευών, αναγνωρίζοντας αυτές που είναι συμβατές με το Weather Station Profile. Στη συνέχεια, συνδέεται μαζί τους και, εάν είναι απαραίτητο, μεταφέρει σε αυτές τις ρυθμίσεις του χρήστη, που είναι αποθηκευμένες τοπικά στο smartphone ή tablet. Από εκεί και πέρα, και με βάση αυτές τις ρυθμίσεις, οι δύο συσκευές επικοινωνούν με τους τρόπους που ορίζονται στο προφίλ. Έτσι, η συσκευή καταγραφής εκτελεί μετρήσεις, τις οποίες στέλνει, στη συνέχεια, στην Android εφαρμογή, ενώ η εφαρμογή, με τη σειρά της, μπορεί να διαβάζει τα αποτελέσματα των τελευταίων μετρήσεων ή να στέλνει εντολές και ρυθμίσεις στη συσκευή καταγραφής.

Σε αυτό το κεφάλαιο περιγράφουμε τη δομή και τον τρόπο λειτουργίας της εφαρμογής που υλοποιήσαμε, τα συστατικά στοιχεία από τα οποία αποτελείται και τους τρόπους αλληλεπίδρασής τους, καθώς και τις δυνατότητες που παρέχονται στον χρήστη μέσα από αυτά. Τέλος, παρουσιάζουμε κάποιες περιπτώσεις χρήσης, με τις οποίες δείχνουμε πώς ένα παρόμοιο σύστημα μπορεί να χρησιμοποιηθεί από κάποιον τελικό χρήστη.

8.1 Γενική περιγραφή της εφαρμογής

Έχουμε ήδη περιγράψει, σε προηγούμενο κεφάλαιο, τα βασικά συστατικά στοιχεία από τα οποία αποτελείται μια Android εφαρμογή. Όπως είδαμε, μια εφαρμογή αποτελείται από μια σειρά από οθόνες, που παρουσιάζουν το user interface της στον χρήστη και αλληλεπιδρούν με αυτόν. Οι οθόνες αυτές ονομάζονται activities. Κάθε activity μπορεί να βρίσκεται σε μία από τις καταστάσεις: resumed, paused ή stopped. Βασικό μέρος της

λειτουργίας του είναι να χειρίζεται γεγονότα που προκύπτουν κατά τη χρήση του, από ενέργειες του χρήστη, όπως η ελαχιστοποίηση της εφαρμογής, η έναρξη άλλων εφαρμογών και η μετακίνηση σε άλλη οθόνη. Η ενημέρωσή τους για αυτά τα γεγονότα γίνεται μέσω μιας σειράς από callback μεθόδους, που καλούνται από το σύστημα σε κάθε περίπτωση. Ο developer ενός Android app πρέπει να υλοποιήσει τον απαραίτητο κώδικα στις μεθόδους των activities που ορίζει, ώστε αυτά να συνεργάζονται αρμονικά με το σύστημα και να παρουσιάζουν την αναμενόμενη από τον χρήστη συμπεριφορά.

Ένα δεύτερο συστατικό στοιχείο, που μπορεί να περιέχεται σε ένα Android app, είναι το service. Ως κύριο σκοπό του έχει την εκτέλεση λειτουργιών της εφαρμογής στο παρασκήνιο, όταν αυτές δε συνδέονται με κάποιο activity (σε αυτήν την περίπτωση θα μπορούσαν να χρησιμοποιηθούν threads) και πρέπει να λειτουργούν ανεξάρτητα από αυτά. Ένα service μπορεί να βρίσκεται σε κάποια από τις καταστάσεις started ή bound, ή και στις δύο, ανάλογα με τον τρόπο εκκίνησης και χρήσης του. Όπως και στην περίπτωση των activities, το σύστημα ειδοποιεί το service για γεγονότα που το επηρεάζουν, μέσω μιας σειράς από callback μεθόδους, τις οποίες πρέπει να υλοποιήσει ο developer.

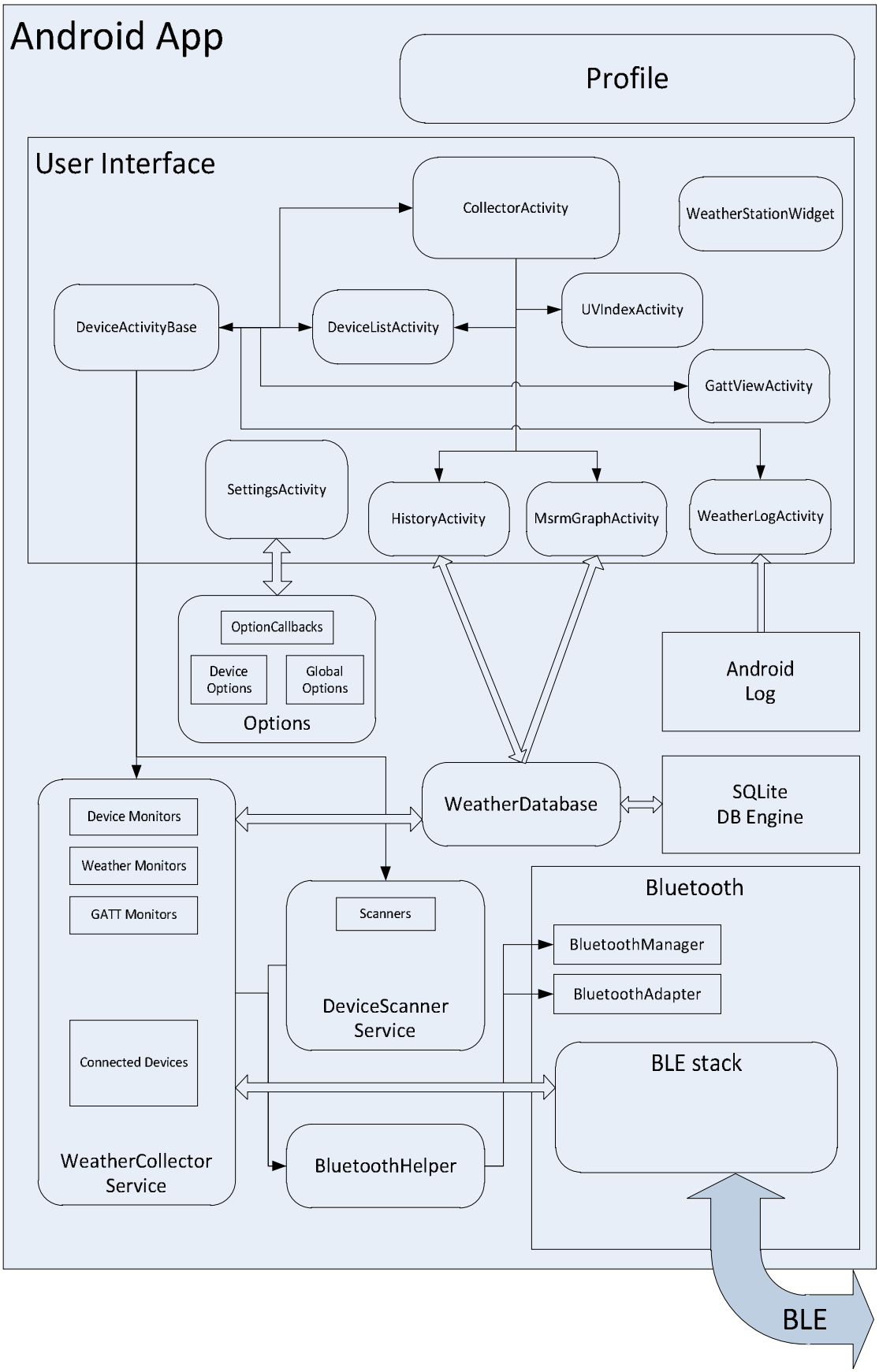
Άλλα συστατικά στοιχεία είναι οι broadcast receivers, που, αφού κάνουν register για συγκεκριμένα system-wide broadcast μηνύματα (intents), μπορούν, στη συνέχεια, να δέχονται από το σύστημα τα μηνύματα αυτά, όταν συμβαίνουν τα αντίστοιχα γεγονότα. Τέλος, μια εφαρμογή έχει τη δυνατότητα ορισμού content providers, μέσω των οποίων μπορεί να παρέχει, σε άλλες εφαρμογές, πρόσβαση σε δεδομένα τα οποία διαχειρίζεται, με δομημένο τρόπο και δυνατότητα ορισμού του απαιτούμενου επίπεδου ασφάλειας.

8.1.1 Συστατικά στοιχεία

Ορίσαμε, λοιπόν, μια σειρά από activities, ένα για κάθε οθόνη της εφαρμογής μας. Επίσης, ορίσαμε services, προκειμένου λειτουργίες, όπως η αναζήτηση συσκευών, η σύνδεση με αυτές και η λήψη των αποτελεσμάτων μετρήσεων, να μπορούν να εκτελεστούν και στο παρασκήνιο, μετά από ρύθμιση του χρήστη. Όπου ήταν απαραίτητο, υλοποιήσαμε και broadcast receivers για τη λήψη μηνυμάτων τόσο από το σύστημα, όσο και από την ίδια την εφαρμογή. Η εφαρμογή μας δεν περιέχει content provider.

Όπως αναφέραμε σε προηγούμενο κεφάλαιο, κατά τον προγραμματισμό για την πλατφόρμα του Android, γίνεται ευρεία χρήση αρχείων XML, μέσω των οποίων ορίζονται πολλά από τα στοιχεία της εφαρμογής. Η εφαρμογή μας περιέχει, λοιπόν, ένα σύνολο από XML αρχεία, μέσω των οποίων ορίζουμε τα strings του UI και το ίδιο το user interface, τόσο για τα activities, όσο και για το widget, το οποίο έχει και ξεχωριστό αρχείο ρυθμίσεων. Επίσης, μέσω XML, ορίζουμε τις ρυθμίσεις που μπορεί να κάνει ο χρήστης, οι οποίες αφορούν τόσο την ίδια την εφαρμογή, όσο και τις συσκευές καταγραφής. Με τη βοήθεια αυτών, και πάλι μέσω XML, ορίζουμε μια σειρά από οθόνες προτιμήσεων, με τις οποίες ο χρήστης μπορεί να ρυθμίσει όλες τις παραμέτρους λειτουργίας του συστήματος. Φυσικά, ανάμεσα στα XML αρχεία, περιέχεται και το, απαραίτητο για όλες τις εφαρμογές, AndroidManifest.xml, όπου δηλώνουμε, ανάμεσα στα άλλα, τα απαραίτητα permissions για τη χρήση του Bluetooth, το API level στο οποίο απευθύνεται η εφαρμογής μας (τουλάχιστον 18, αφού από αυτή την έκδοση ξεκίνησε η υποστήριξη του BLE), και, βέβαια, δηλώνουμε όλα τα components που απαρτίζουν την εφαρμογή μας.

Τα βασικότερα συστατικά στοιχεία της εφαρμογής, καθώς και οι κυριότερες αλληλεπιδράσεις τους, παρουσιάζονται στο διάγραμμα της διπλανής σελίδας. Αφού κάνουμε εδώ μια συνοπτική περιγραφή τους, θα ακολουθήσει λεπτομερής ανάλυση της λειτουργίας τους.



Δομή της εφαρμογής Android.

Η κλάση Profile είναι μια πολύ σημαντική κλάση της εφαρμογής, αφού περιέχει ορισμούς και δεδομένα που σχετίζονται άμεσα με το weather station profile. Περιέχει όλα τα UUIDs για υπηρεσίες, χαρακτηριστικά και descriptors, που ορίζονται στο προφίλ, και πρέπει να υποστηρίζονται από τις συμβατές συσκευές. Επίσης, περιέχει μια σειρά από βοηθητικές κλάσεις, δεδομένα και μεθόδους για διάφορες λειτουργίες που προβλέπονται στο προφίλ. Χρησιμοποιείται σχεδόν σε όλα τα στοιχεία της εφαρμογής.

Η κλάση Options ελέγχει κεντρικά όλες τις ρυθμίσεις που μπορούν να γίνουν στην εφαρμογή. Οι ρυθμίσεις αυτές πραγματοποιούνται μέσω κατάλληλων UIs, που εμφανίζονται στον χρήστη με τη βοήθεια του SettingsActivity και περιεχόμενων σε αυτό κλάσεων. Οι ρυθμίσεις αφορούν τόσο τη λειτουργία της ίδιας της εφαρμογής, όσο και τις παραμέτρους λειτουργίας των συσκευών καταγραφής, τις οποίες έχουμε δει στα προηγούμενα κεφάλαια. Όταν συμβαίνει κάποια αλλαγή στις ρυθμίσεις, η κλάση Options ενημερώνει όλα τα ενδιαφερόμενα στοιχεία της εφαρμογής.

Ο BluetoothHelper είναι μια βοηθητική κλάση, που χρησιμοποιείται από τα στοιχεία της εφαρμογής, τα οποία χρησιμοποιούν το Bluetooth hardware. Ο BluetoothHelper απλοποιεί κάποιες από τις διαδικασίες που απαιτούνται, προκειμένου η εφαρμογή να αποκτήσει πρόσβαση στο Bluetooth. Δίνει τη δυνατότητα ελέγχου της παρουσίας του απαραίτητου hardware, και ενημέρωσης, σε περίπτωση που συμβεί κάποια αλλαγή στην κατάστασή του.

Για την αποθήκευση των αποτελεσμάτων των μετρήσεων ορίσαμε μια απλή βάση δεδομένων με ένα table ανά μέγεθος. Για την υλοποίησή της, χρησιμοποιήθηκε η μηχανή βάσης δεδομένων SQLite που παρέχεται από το Android. Η πρόσβαση στη βάση δεδομένων ελέγχεται από την κλάση WeatherDatabase, την οποία χρησιμοποιούν τα υπόλοιπα στοιχεία για να προσθέσουν, να ανακτήσουν ή να διαγράψουν δεδομένα από αυτή.

Η εφαρμογή περιέχει δύο services, τα DeviceScanner και WeatherCollector service. Ο DeviceScanner εκτελεί την αναζήτηση συσκευών και ενημερώνει τα ενδιαφερόμενα στοιχεία της εφαρμογής, όταν βρεθεί μια συσκευή καταγραφής καιρού. Η αναζήτηση μπορεί να γίνεται στο παρασκήνιο, ακόμα και αν υπάρχει ήδη σύνδεση με μια συσκευή, ή μπορεί να εκκινείται και να τερματίζεται μετά από εντολή του χρήστη. Ο WeatherCollector είναι η σημαντικότερη κλάση της εφαρμογής. Αποτελεί το κέντρο όλου του συστήματος και σε αυτόν περιέχεται το μεγαλύτερο μέρος της λειτουργικότητας και της συμπεριφοράς που προβλέπονται από το weather station profile. Πραγματοποιεί τις συνδέσεις με τις συσκευές που ανακαλύπτονται από τον DeviceScanner και εκτελεί όλη την επικοινωνία που απαιτείται, προκειμένου να υλοποιηθεί η λειτουργία του συστήματος καταγραφής καιρού, όπως ορίζεται στο προφίλ. Ενημερώνει τα ενδιαφερόμενα στοιχεία της εφαρμογής για διάφορα γεγονότα, όπως σύνδεση, αποσύνδεση και λήψη νέας μέτρησης. Επίσης, οι DeviceScanner και WeatherCollector παρέχουν μια σειρά από υπηρεσίες που σχετίζονται με τη λειτουργία τους. Τα υπόλοιπα στοιχεία της εφαρμογής, που θέλουν να χρησιμοποιήσουν τις υπηρεσίες αυτές, κάνουν bind με το αντίστοιχο service. Όπως αναφέραμε παραπάνω, είναι δυνατόν οι λειτουργίες των services να εκτελούνται και στο παρασκήνιο. Σε αυτήν την περίπτωση, τα services είναι και σε κατάσταση started.

Η κλάση DeviceActivityBase είναι η βασική κλάση activity, μέσω της οποίας ορίζονται όλα τα υπόλοιπα activities που χρησιμοποιούν τις υπηρεσίες των DeviceScanner και WeatherCollector. Η κλάση DeviceActivityBase αναλαμβάνει και αυτοματοποιεί πολλές από τις διαδικασίες που απαιτούνται, προκειμένου να γίνει η σύνδεση με τα services, δίνοντας, παράλληλα, στα activities που δημιουργούνται μέσω αυτής, πλήρη δυνατότητα παραμετροποίησης της λειτουργίας της. Κατά την υλοποίηση της εφαρμογής ορίσαμε τέσσερα τέτοια activities ως υποκλάσεις της DeviceActivityBase, τα WeatherLogActivity, GattViewActivity, DeviceListActivity και CollectorActivity.

Τα WeatherLogActivity και GattViewActivity είναι βοηθητικά εργαλεία για τον έλεγχο της λειτουργίας της εφαρμογής. Το WeatherLogActivity παρέχει ένα βασικό interface, μέσω του οποίου παρουσιάζει τα αποτελέσματα των μετρήσεων που λαμβάνονται, δίνοντας επίσης τη δυνατότητα άμεσης ρύθμισης παραμέτρων λειτουργίας από την αρχική οθόνη. Ταυτόχρονα, συνδέεται με το σύστημα log της πλατφόρμας και εμφανίζει το debug log της εφαρμογής. Το GattViewActivity παρουσιάζει, μέσω κατάλληλου UI, τις υπηρεσίες και τα χαρακτηριστικά που περιέχονται στη συσκευή καταγραφής. Δίνει τη δυνατότητα ανάκτησης και εγγραφής των τιμών των χαρακτηριστικών στο επίπεδο του GATT, καθώς και τη δυνατότητα ρύθμισης των cliect configuration descriptors.

Το DeviceListActivity παρέχει μια λίστα γνωστών συσκευών καταγραφής, είτε συνδεμένων, είτε αποσυνδεμένων. Δίνει, επίσης, τη δυνατότητα εκκίνησης αναζήτησης, παρουσιάζοντας τις συσκευές που βρέθηκαν. Στη συνέχεια, ο χρήστης μπορεί να συνδεθεί με κάποια από αυτές. Τέλος, το CollectorActivity είναι το βασικό user interface του συστήματος καταγραφής καιρού. Παρουσιάζει τα αποτελέσματα των μετρήσεων στον χρήστη και παρέχει πρόσβαση σε όλες τις άλλες λειτουργίες και activities της εφαρμογής. Τέτοια activities είναι τα HistoryActivity, MsrmGraphActivity και UVIndexActivity. Το HistoryActivity παρέχει το ιστορικό των μετρήσεων. Το MsrmGraphActivity δίνει τη δυνατότητα παρουσίασης των μετρήσεων μέσω γραφικής παράστασης. Το UVIndexActivity παρέχει πληροφορίες σχετικά με τα μέτρα που πρέπει να ληφθούν για την προστασία από την υπεριώδη ακτινοβολία, με βάση την τρέχουσα τιμή της, όπως αυτή μετρείται από τη συσκευή καταγραφής.

Τέλος, ορίσαμε ένα απλό app widget, με την κλάση WeatherStationWidget, το οποίο ενημερώνεται για τα αποτελέσματα των μετρήσεων από τον WeatherCollector, και τα παρουσιάζει στη home screen της συσκευής.

8.1.2 Ρόλοι της Android εφαρμογής

Όπως κάναμε στο προηγούμενο κεφάλαιο, έτσι κι εδώ, αναφέρουμε τους ρόλους με τους οποίους η Android εφαρμογή, και η συσκευή στην οποία εκτελείται, εμφανίζονται στα διάφορα επίπεδα της αρχιτεκτονικής του συστήματος. Πιο συγκεκριμένα, έχουμε:

- **GAP**: Στο επίπεδο του GAP, η Android εφαρμογή είναι central.
- **GATT**: Στο επίπεδο του GATT, η Android εφαρμογή είναι GATT client.
- **ATT**: Στο επίπεδο του ATT, ισχύει ό,τι και στο επίπεδο του GATT, οπότε η Android εφαρμογή είναι ATT client.
- **Link Layer**: Στο Link Layer η Android εφαρμογή λειτουργεί ως scanner και initiator και, όταν δημιουργηθεί σύνδεση με κάποια απομακρυσμένη συσκευή, μετέχει στη σύνδεση με το ρόλο του master.
- **Weather Station Profile**: Στο weather station profile η Android εφαρμογή έχει το ρόλο του weather collector.
- **Τύπος Συσκευής**: Ο τύπος της συσκευής, με βάση το διαχωρισμό που έχει ορίσει το SIG, είναι Bluetooth Smart Ready. Δηλαδή dual-mode συσκευή με δυνατότητα αναβάθμισης, η οποία υποστηρίζει το BLE και μπορεί να συνδέεται με Bluetooth Smart συσκευές, παρέχοντας λειτουργικότητα μέσω του GATT.

8.1.3 Χρήση Singleton Design Pattern

Κατά την υλοποίηση των παραπάνω, χρησιμοποιήθηκε σε αρκετές περιπτώσεις το singleton design pattern.

```
private static Singleton singleton;

private Singleton () {
...
}

public static Singleton getInstance () {
    if (singleton == null)
        singleton = new Singleton();
    return singleton;
}
```

Ο λόγος που ακολουθήσαμε την παραπάνω σχεδίαση είναι ότι πολλά από τα στοιχεία της εφαρμογής ελέγχουν την πρόσβαση, ή εξαρτώνται από τμήματα του συστήματος και της εφαρμογής, τα οποία είναι μοναδικά, όπως το Bluetooth hardware, η βάση δεδομένων και τα αρχεία ρυθμίσεων. Οπότε, τα αντίστοιχα στοιχεία υλοποιούνται ως singleton, ώστε και αυτά να είναι μοναδικά στην εφαρμογή. Τέτοιες περιπτώσεις είναι τα BluetoothHelper, WeatherDatabase και Options, τα οποία, όπως προβλέπεται στο design pattern, δημιουργούνται μία φορά κατά την πρώτη χρήση τους και, από εκεί και πέρα, χρησιμοποιείται το ίδιο αντικείμενο. Επίσης, αξίζει εδώ να σημειωθεί ότι μερικά από τα components της εφαρμογής μας, αν και δεν ορίζονται ως singleton, χρησιμοποιούνται από το σύστημα ως τέτοια. Για παράδειγμα ένα service δημιουργείται, όταν απαιτείται η χρήση του, μόνο αν δεν υπάρχει ήδη. Αν υπάρχει, χρησιμοποιείται το ήδη υπάρχον instance. Έτσι, αντικείμενα όπως τα services WeatherCollector και DeviceScanner είναι επίσης μοναδικά στην εφαρμογή.

8.1.4 Android Manifest

Το AndroidManifest.xml ξεκινά με τη δήλωση του API level και των, απαραίτητων για τη χρήση του Bluetooth, permissions. Επίσης, δηλώνουμε ότι η εφαρμογή μας απαιτεί την υποστήριξη Bluetooth Low Energy.

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="gr.ntua.microlab.weatherstation"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="18"
        android:targetSdkVersion="19" />

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <uses-feature
        android:name="android.hardware.bluetooth_le"
        android:required="true" />
...
```

Στη συνέχεια, δηλώνουμε όλα τα στοιχεία της εφαρμογής μας, προκειμένου να μπορούν να χρησιμοποιηθούν.

```
<application
    android:name="gr.ntua.microlab.weatherstation.App"
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
        android:name="gr.ntua.microlab.weatherstation.CollectorActivity"
        android:label="@string/title_activity_collector"
        android:configChanges="orientation|screenSize"
        android:screenOrientation="portrait" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <activity
        android:name="gr.ntua.microlab.weatherstation.SettingsActivity"
        android:label="@string/title_activity_settings" >
    </activity>

    <activity
        android:name="gr.ntua.microlab.weatherstation.DeviceListActivity"
        android:label="@string/title_activity_device_list"
        android:configChanges="orientation|screenSize" >
    </activity>

...
<service android:name="gr.ntua.microlab.weatherstation.DeviceScanner" />
<service android:name="gr.ntua.microlab.weatherstation.WeatherCollector" />
...
```

Για τον WeatherCollector, ορίζουμε επίσης ένα broadcast receiver, μέσω του οποίου μπορεί να ενημερώνεται από τον DeviceScanner για την εύρεση συσκευών, ώστε να συνδεθεί αυτόματα με αυτές, αν η καταγραφή στο παρασκήνιο είναι ενεργοποιημένη.

```
<receiver
    android:name=
    "gr.ntua.microlab.weatherstation.WeatherCollector$DeviceFoundReceiver"
    android:exported="false" >
    <intent-filter>
       <action
       android:name="gr.ntua.microlab.weatherstation.action.DEVICE_FOUND" />
    </intent-filter>
</receiver>
```

Τέλος, δηλώνουμε και το app widget. Το app widget είναι ένας broadcast receiver. Εκτός του intent που είναι απαραίτητο για τη λειτουργία του, κάνουμε register και για intents που ορίζονται από τον WeatherCollector και χρησιμοποιούνται για broadcast των μετρήσεων που λαμβάνονται.

```
<receiver
    android:name="gr.ntua.microlab.weatherstation.WeatherStationWidget" >
    <intent-filter>
        <action
            android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/weather_station_widget_info" />

    <intent-filter>
        <action
```

```
android:name="gr.ntua.microlab.weatherstation.action.MEASUREMENT" />
    </intent-filter>
    <intent-filter>
        <action
android:name="gr.ntua.microlab.weatherstation.action.DEVICE_CONNECTED" />
    </intent-filter>
    <intent-filter>
        <action
android:name="gr.ntua.microlab.weatherstation.action.DEVICE_DISCONNECTED" />
    </intent-filter>
</receiver>
```

8.2 Βοηθητικά στοιχεία

Σε αυτή την υπο-ενότητα αναφέρουμε ορισμένα βοηθητικά στοιχεία που υλοποιήσαμε.

8.2.1 Debug Log

Όπως και στην περίπτωση της ενσωματωμένης εφαρμογής, που εκτελείται στο DA14580, έτσι και στην Android εφαρμογή, αποφασίσαμε να υλοποιήσουμε ένα debug log για τον έλεγχο της λειτουργίας και της γενικότερης συμπεριφοράς της. Το debug log υπήρξε και σε αυτή την περίπτωση ιδιαίτερα χρήσιμο για την ανακάλυψη και επίλυση προβλημάτων της εφαρμογής. Η υλοποίησή του στο Android ήταν ευκολότερη από ό,τι στο DA14580, αφού το Android παρέχει κατάλληλο API για τη δημιουργία debug logs. Για την ακρίβεια, δίνεται από το σύστημα του Android η δυνατότητα στις εφαρμογές, να γράφουν μηνύματα στο γενικό log του συστήματος, χρησιμοποιώντας tags και καθορίζοντας το επίπεδο σπουδαιότητας του μηνύματος (assert, error, warn, info, debug, verbose, με φθίνουσα σημασία). Τα μηνύματα αυτά μπορούν να διαβαστούν μέσω μιας εφαρμογής που περιέχεται στο Android, η οποία ονομάζεται logcat. Το ADT δίνει τη δυνατότητα στους developers να βλέπουν το log του συστήματος και της εφαρμογής τους κατά το debugging, καθώς και να ορίζουν φίλτρα για την επιλογή συγκεκριμένων μηνυμάτων που τους ενδιαφέρουν.

Για τη δημιουργία του debug log, ορίσαμε μια σειρά από βοηθητικές μεθόδους, που καλούνται σε διάφορα σημεία της λειτουργίας της εφαρμογής, παράγοντας έξοδο που ενημερώνει για την εξέλιξή της. Υλοποιήσαμε, επίσης, δυνατότητα παρακολούθησης του debug log, κατά τη διάρκεια της χρήσης της εφαρμογής, η οποία χρησιμοποιείται από το WeatherLogActivity. Όπως και στην ενσωματωμένη εφαρμογή, δίνεται η δυνατότητα απενεργοποίησης του log, για τη μείωση του μεγέθους του κώδικα και, κυρίως, την αύξηση της απόδοσης της εφαρμογής.

8.2.2 NameResolver

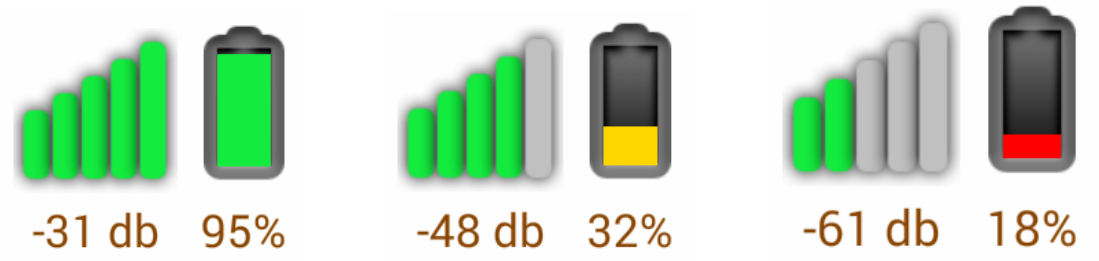
Ο NameResolver περιέχει μια αντιστοίχηση (map) μεταξύ γνωστών UUIDs υπηρεσιών, χαρακτηριστικών και descriptors, και των ονομάτων τους. Χρησιμοποιείται από το debug log της εφαρμογής, στις εγγραφές στο log που αφορούν συγκεκριμένα GATT αντικείμενα, για τη μετατροπή του UUID του αντικειμένου στο αντίστοιχο όνομα. Με αυτό τον τρόπο το log καθίσταται περισσότερο ευανάγνωστο. Επίσης, χρησιμοποιείται από το GattViewActivity για τη δημιουργία της λίστας των υπηρεσιών και των χαρακτηριστικών.

8.2.3 Uint24

Το weather station profile χρησιμοποιεί το format uint24 του BLE για την αποθήκευση τιμών διάφορων χαρακτηριστικών που ορίζονται σε αυτό. Τα measurement intervals σε όλα τα services έχουν το παραπάνω format, όπως επίσης και οι μετρήσεις της ατμοσφαιρικής πίεσης και του ambient light. Το BLE API του Android παρέχει κάποιες μεθόδους για τη μετατροπή μεταξύ τύπων δεδομένων της Java και format του BLE. Για παράδειγμα, είναι δυνατή η μετατροπή της τιμής ενός χαρακτηριστικού μεταξύ Java float και BLE SFLOAT. Δε συμβαίνει όμως το ίδιο με το uint24 format. Για αυτό τον λόγο υλοποιήσαμε μια απλή κλάση, η οποία μπορεί να μετατρέπει τιμές μεταξύ Java int και BLE uint24. Χρησιμοποιείται όπου είναι απαραίτητη αυτή η μετατροπή, όπως, για παράδειγμα, κατά την αλλαγή ρύθμισης των περιοδικών μετρήσεων, στην οποία απαιτείται η εγγραφή των τοπικών τιμών, που είναι αποθηκευμένες σε μεταβλητές τύπου int, στα χαρακτηριστικά measurement interval των services της συσκευής καταγραφής.

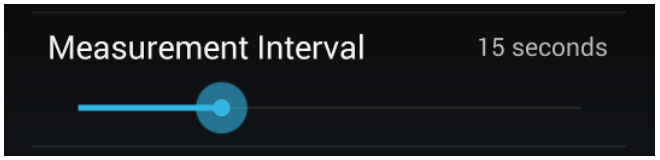
8.2.4 BatteryImageView – RssiImageView

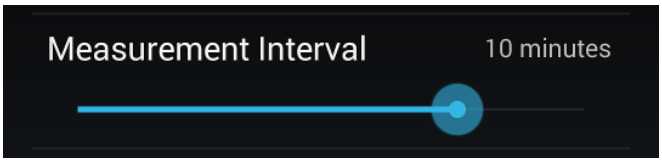
Οι κλάσεις BatteryImageView και RssiImageView είναι υποκλάσεις της ImageView, η οποία παρέχεται από το Android API για την τοποθέτηση εικόνων στο UI. Τα αντικείμενα των συγκεκριμένων κλάσεων συνδυάζουν διάφορα αντικείμενα που παρέχονται από το API, προκειμένου να δημιουργήσουν γραφικά που παρουσιάζουν στον χρήστη την τιμή του αντίστοιχου μεγέθους, δηλαδή, το επίπεδο της μπαταρίας της συσκευής καταγραφής για το BatteryImageView, και το επίπεδο του λαμβανόμενου από αυτή σήματος για το RssiImageView. Και οι δύο παρέχουν κατάλληλη μέθοδο με την οποία η εφαρμογή μπορεί να αλλάξει την τιμή του αντίστοιχου μεγέθους. Έτσι, κάθε φορά που διαβάζεται το επίπεδο της μπαταρίας της συσκευής καταγραφής, το αντίστοιχο γραφικό στο UI ανανεώνεται με κλήση της μεθόδου setBatteryLevel του BatteryImageView. Η ενεργοποίηση της περιοδικής ανάκτησης του επιπέδου της μπαταρίας, καθώς και η περίοδος αυτής, ρυθμίζονται από τον χρήστη.



8.2.5 MeasurementIntervalPreference

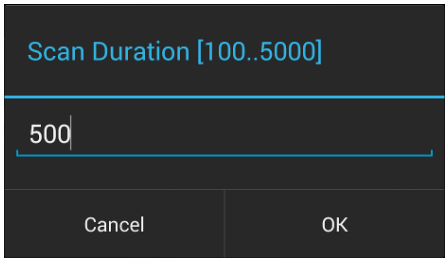
Η κλάση MeasurementIntervalPreference χρησιμοποιείται για τον ορισμό του measurement interval ενός service, μέσω ενός seek bar widget, το οποίο περιέχεται απευθείας στην οθόνη ρυθμίσεων και όχι σε dialog. Η υλοποίησή της έγινε, επειδή δεν προσφέρεται κάτι αντίστοιχο στο Android API. Ο χρήστης έχει τη δυνατότητα να επιλέξει μεταξύ κάποιων προκαθορισμένων τιμών από 1 δευτερόλεπτο μέχρι 1 ώρα. Εμφανίζεται, επίσης, η τρέχουσα ρύθμιση.

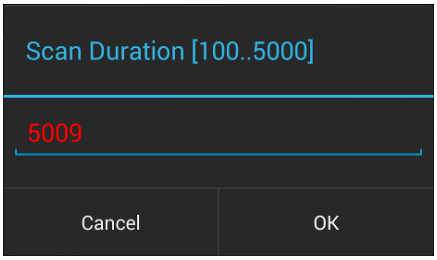




8.2.6 EditNumberPreference

Η κλάση EditNumberPreference χρησιμοποιείται για την εισαγωγή αριθμών από τον χρήστη, σε ρυθμίσεις όπου απαιτείται κάτι τέτοιο. Η διαφορά της από το αντίστοιχο UI του Android API είναι ότι επιτρέπει, επιπλέον, τον καθορισμό ορίων στις δυνατές τιμές εισόδου. Αν ο χρήστης εισάγει μία τιμή εκτός ορίων, λαμβάνει οπτική προειδοποίηση. Σε κάθε περίπτωση, το EditNumberPreference δεν επιτρέπει την αποθήκευση στις ρυθμίσεις, τιμών εκτός ορίων. Αν ο μία τιμή είναι εκτός ορίων, θέτει την αντίστοιχη ρύθμιση στη μέγιστη ή ελάχιστη τιμή.





8.3 Profile

Η κλάση Profile περιέχει όλα τα δεδομένα που ορίζονται από το weather station profile, τα οποία είναι απαραίτητα για την υλοποίηση της συμπεριφοράς, που προβλέπεται από αυτό για συμβατές συσκευές. Πιο συγκεκριμένα, περιλαμβάνει:

- UUIDs υπηρεσιών, χαρακτηριστικών και descriptors για όλα τα αντικείμενα GATT που είναι απαραίτητα. Αυτά περιλαμβάνουν το master service και όλα τα sensor services, καθώς και τα SIG UUIDs για services που χρησιμοποιούνται, όπως το Battery Service, το Device Information Service, το GAP και το GATT service.

- SIG Unit UUIDs, τα οποία ορίζονται στα Assigned Numbers του SIG, και χρησιμοποιούνται από τα Presentation Formats των χαρακτηριστικών measurement.
- Σημαίες που ορίζονται στο προφίλ για το master service measurement, οι οποίες δηλώνουν την ύπαρξη συγκεκριμένων μετρήσεων, στα δεδομένα που στέλνονται μέσω αυτού, από τη συσκευή καταγραφής. Περιέχονται, επίσης, και τα μεγέθη των τιμών των διαφόρων μετρήσεων. Χρησιμοποιούνται για τον διαχωρισμό των πακεταρισμένων δεδομένων.
- Τα sensor IDs που ορίζονται στο προφίλ. Χρησιμοποιούνται για την αποστολή εντολών στο master service control point.
- Τις τιμές των opcodes των εντολών, καθώς και τις τιμές που επιστρέφονται από τη συσκευή καταγραφής.
- Default τιμές για διάφορες ρυθμίσεις που χρησιμοποιούνται στην εφαρμογή.

Τα UUIDs ορίζονται, όπως και στην ενσωματωμένη εφαρμογή, με βάση τη διαφορά τους από το βασικό UUID. Με τον ίδιο τρόπο ορίζονται και τα SIG UUIDs.

```
...

public static class WthrsUUID {
    public static final UUID COMMON =
                    UUID.fromString("dc980000-f292-11e3-b75f-002215f5ef22");

    public static final UUID get(int diff) {
        return BTUUID.common(diff, COMMON);
    }
}

// Weather station UUID differentiate part
public static class Diff {
    public static final int WEATHER_STATION     = 0x1000;
    public static final int TEMPERATURE         = 0x1100;
    public static final int HUMIDITY            = 0x1200;
    ...

    public static final int WTHRS_MSRM          = WEATHER_STATION + 1;
    public static final int WTHRS_MSRM_INT      = WEATHER_STATION + 2;
    public static final int WTHRS_CTRL_PT       = WEATHER_STATION + 3;

    public static final int TEMP_MSRM           = TEMPERATURE + 1;
    public static final int TEMP_MSRM_INT       = TEMPERATURE + 2;
    public static final int TEMP_CTRL_PT        = TEMPERATURE + 3;

    public static final int HUMID_MSRM          = HUMIDITY + 1;
    public static final int HUMID_MSRM_INT      = HUMIDITY + 2;
    public static final int HUMID_CTRL_PT       = HUMIDITY + 3;
    ...
}

public static class Service {
    // Weather station services
    public static final UUID WEATHER_STATION =
                                    WthrsUUID.get(Diff.WEATHER_STATION);
    public static final UUID TEMPERATURE = WthrsUUID.get(Diff.TEMPERATURE);
    public static final UUID HUMIDITY    = WthrsUUID.get(Diff.HUMIDITY);
    ...
    // BLE services
    public static final UUID GAP                = BTUUID.sig(0x1800);
    public static final UUID GATT               = BTUUID.sig(0x1801);
    public static final UUID BATTERY            = BTUUID.sig(0x180F);
    public static final UUID DEVICE_INFORMATION = BTUUID.sig(0x180A);
}

...
```

Η κλάση Profile παρέχει, επίσης, μια σειρά από βοηθητικές μεθόδους και κλάσεις, που υλοποιούν γενικές λειτουργίες του BLE ή του weather station profile. Η σημαντικότερη είναι η κλάση Measurement, η οποία παρέχει μέθοδο διαχωρισμού των μετρήσεων, που στέλνονται πακεταρισμένες μέσω του master service. Η Measurement χρησιμοποιείται γενικότερα στην εφαρμογή, για την αναπαράσταση των μετρήσεων ενός ή περισσότερων μεγεθών.

8.4 Options

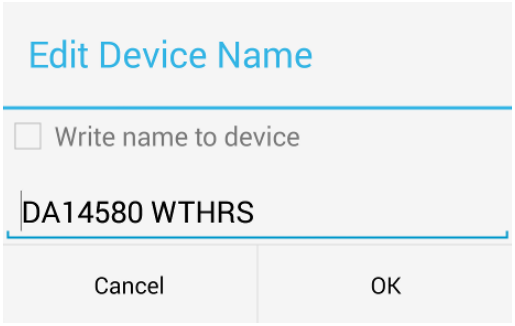
Η κλάση Options χρησιμοποιείται για τον κεντρικό χειρισμό των ρυθμίσεων, τόσο της εφαρμογής, όσο και των συσκευών καταγραφής. Υλοποιείται ως singleton. Για τον ορισμό και την αποθήκευση των ρυθμίσεων χρησιμοποιείται το SharedPreferences API του Android.

8.4.1 Ρυθμίσεις συσκευών καταγραφής

Κάθε συσκευή καταγραφής μπορεί να έχει διαφορετικές ρυθμίσεις, οι οποίες αποθηκεύονται σε διαφορετικό αρχείο και φορτώνονται όταν χρειάζονται. Δίνεται, επίσης, η δυνατότητα στον χρήστη να ορίσει default τιμές ρυθμίσεων για νέες συσκευές. Οι ρυθμίσεις που μπορούν να γίνουν για κάθε συσκευή περιλαμβάνουν όλες τις παραμέτρους λειτουργίας που ορίζονται στο προφίλ, και έχουν περιγραφεί στα προηγούμενα κεφάλαια. Σε αυτήν την υπο-ενότητα περιγράφεται ο τρόπος με τον οποίο πραγματοποιούνται οι ρυθμίσεις, μέσω της Android εφαρμογής.

8.4.1.1 Τοπικό όνομα συσκευής

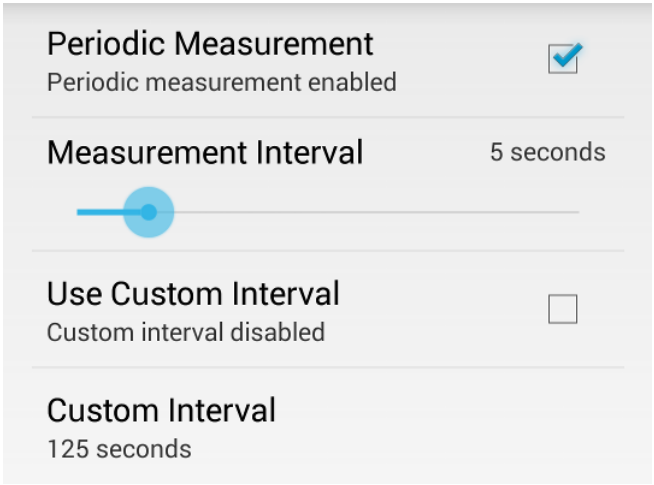
Ο χρήστης έχει τη δυνατότητα ορισμού ενός ονόματος για τη συσκευή καταγραφής, διαφορετικού από αυτό που η ίδια χρησιμοποιεί. Αν επιτρέπεται από τη συσκευή, τότε είναι δυνατόν να γραφτεί το GAP name της. Σε κάθε περίπτωση, μετά τη ρύθμιση, η συσκευή εμφανίζεται στην εφαρμογή με το νέο της όνομα.



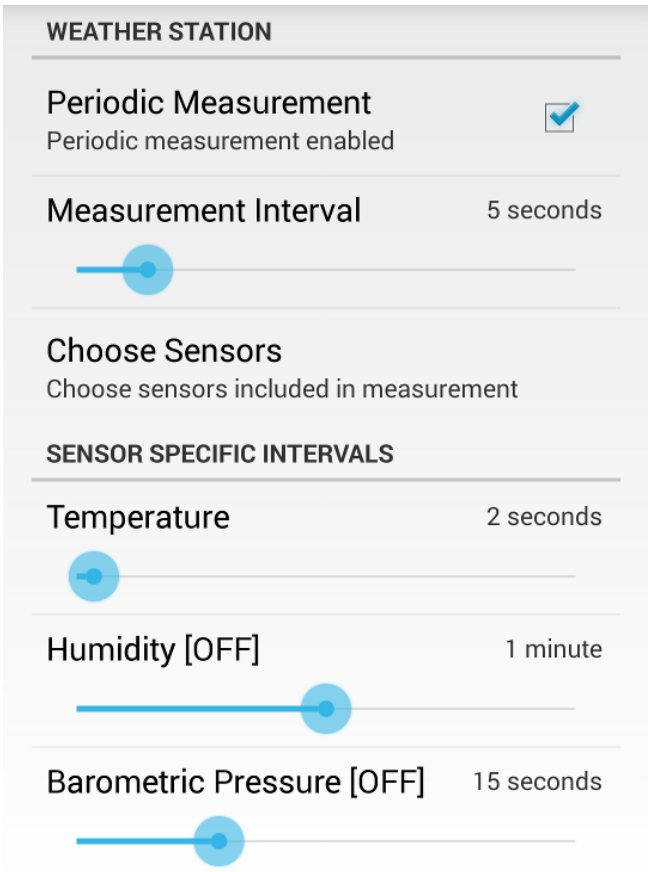
8.4.1.2 Measurement Intervals

Ο χρήστης μπορεί να ρυθμίσει την τιμή της περιόδου των περιοδικών μετρήσεων, είτε σε μια από τις προκαθορισμένες τιμές, που παρέχονται μέσω του seek bar, είτε σε κάποια custom τιμή (μέχρι 86400 δευτερόλεπτα, δηλαδή το μέγιστο που προβλέπει το προφίλ). Δε δίνεται η δυνατότητα να θέσει την τιμή σε μηδέν. Αντίθετα παρέχεται ξεχωριστή ρύθμιση για την απενεργοποίηση των περιοδικών μετρήσεων. Εδώ υπάρχει μια ασυμμετρία μεταξύ των

δύο τμημάτων του συστήματος. Στη συσκευή καταγραφής, η συγκεκριμένη ρύθμιση είναι μοναδική για κάθε service, ενώ στο Android app υπάρχουν πολλές διαφορετικές ρυθμίσεις. Η εφαρμογή θυμάται όλες αυτές τις ρυθμίσεις. Για παράδειγμα, το custom interval και η θέση του seek bar διατηρούνται ακόμα και αν ο χρήστης έχει απενεργοποιήσει τις περιοδικές μετρήσεις. Όταν όμως ο χρήστης αλλάζει οποιαδήποτε από τις ρυθμίσεις αυτές, οι τρέχουσες τιμές τους συνδυάζονται, ώστε να παραχθεί η τιμή που πρέπει τελικά να γραφτεί στη συσκευή καταγραφής.

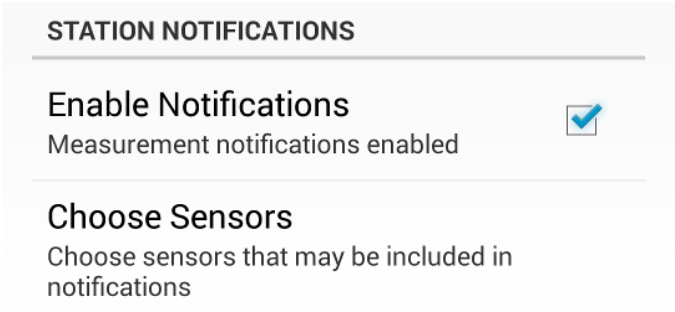


Επειδή η ρύθμιση αυτή αναμένεται να είναι από τις πιο συνηθισμένες που εκτελούνται από τον χρήστη, παρέχεται και σε ξεχωριστή οθόνη, όπου μπορούν να τεθούν όλα τα intervals. Στην οθόνη αυτή, παρέχεται, για τα sensor services, οπτική επιβεβαίωση για την ενεργοποίηση της περιοδικής τους μέτρησης, η οποία, όπως προβλέπεται στο προφίλ, έχει προτεραιότητα σε σχέση με αυτή του master service.

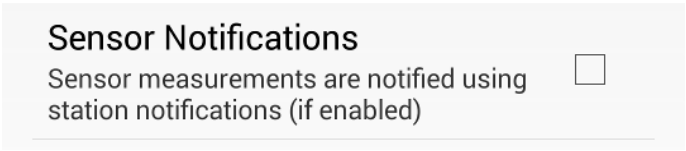


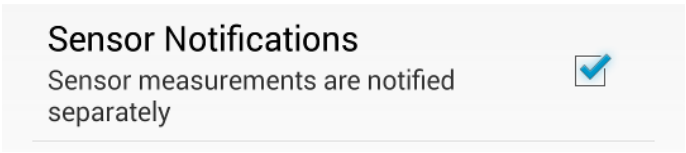
8.4.1.3 Αποστολή αποτελεσμάτων

Ο χρήστης μπορεί να ρυθμίσει τις αποστολές των αποτελεσμάτων για όλα τα services. Η αλλαγή των ρυθμίσεων αυτών, μεταφράζεται σε μεταβολή της τιμής του αντίστοιχου client configuration, για την ενεργοποίηση ή απενεργοποίηση των notifications.



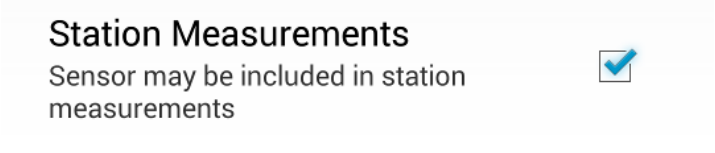
Η παραπάνω οθόνη αφορά το master service. Για κάθε αισθητήρα υπάρχει διαφορετική ρύθμιση, που παρέχεται από την οθόνη ρυθμίσεων του.

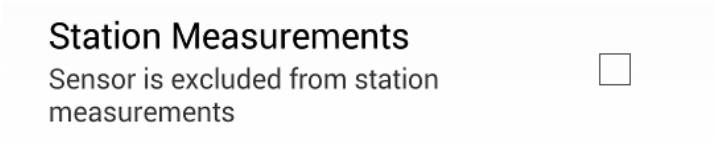




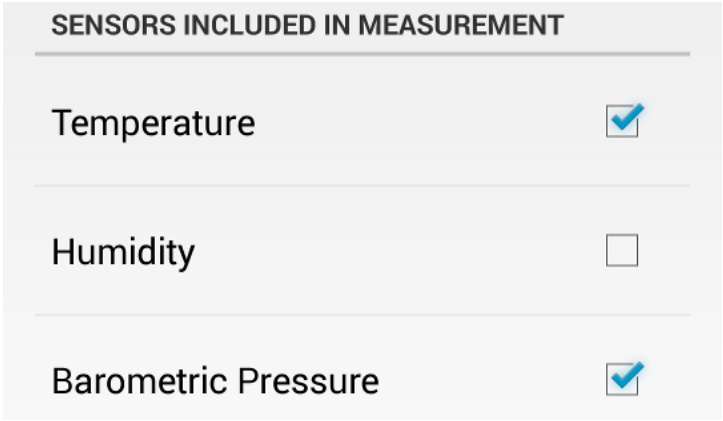
8.4.1.4 Μετρήσεις του master service

Ο χρήστης μπορεί να καθορίσει ποιες μετρήσεις μπορούν να εκκινούνται από το master service. Για κάθε αισθητήρα υπάρχει διαφορετική ρύθμιση, που παρέχεται από την οθόνη ρυθμίσεων του.



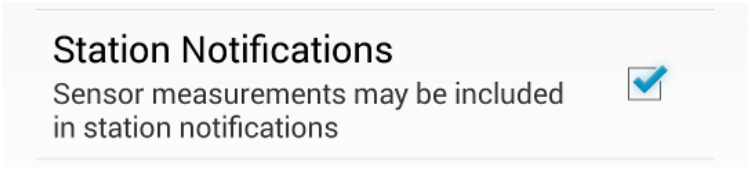


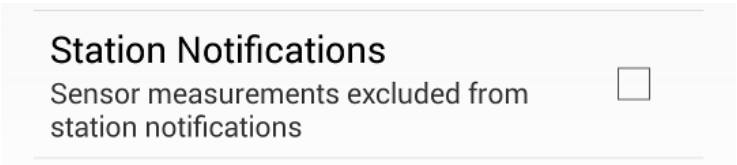
Υπάρχει επίσης και κεντρική ρύθμιση, στην οθόνη ρυθμίσεων του master service measurement.



8.4.1.5 Αποστολές μέσω του master service

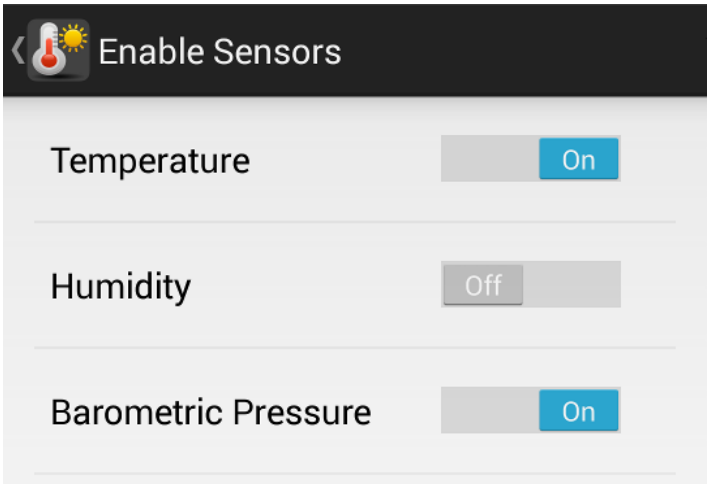
Ο χρήστης μπορεί να καθορίσει ποιες μετρήσεις μπορούν να στέλνονται πακεταρισμένες, μέσω του master service measurement. Για κάθε αισθητήρα υπάρχει διαφορετική ρύθμιση, που παρέχεται από την οθόνη ρυθμίσεων του. Επίσης, όπως και πριν, υπάρχει και κεντρική ρύθμιση, στην οθόνη ρυθμίσεων του master service measurement.





8.4.1.6 Ενεργοποίηση αισθητήρων

Ο χρήστης έχει τη δυνατότητα να ενεργοποιήσει ή να απενεργοποιήσει αισθητήρες, τόσο μέσα από τις ξεχωριστές οθόνες ρυθμίσεών τους, όσο και κεντρικά από κατάλληλη οθόνη που παρέχεται.



8.4.1.7 Αποστολή ρυθμίσεων

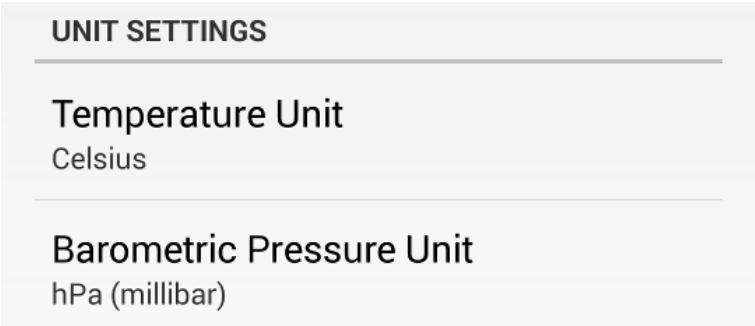
Οι παραπάνω ρυθμίσεις μεταφράζονται από τον WeatherCollector σε εγγραφές στα κατάλληλα χαρακτηριστικά του weather station, και στέλνονται στον τελευταίο με κάθε αλλαγή. Η εφαρμογή ελέγχει αν όντως έχει αλλάξει η ρύθμιση, ώστε να μην εκτελείται επικοινωνία χωρίς να είναι απαραίτητο.

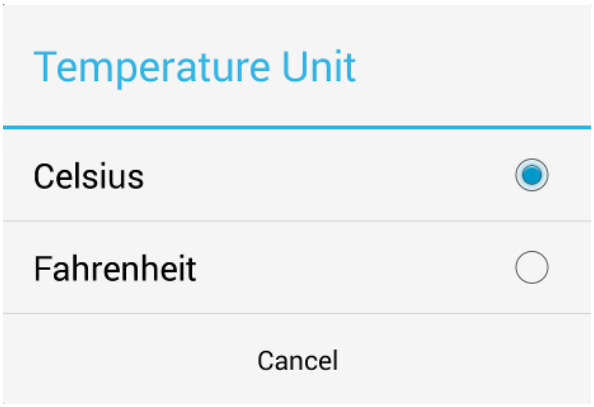
8.4.2 Τοπικές ρυθμίσεις

Πέρα από τις ρυθμίσεις των απομακρυσμένων συσκευών, υπάρχουν και οι ρυθμίσεις της ίδιας της εφαρμογής. Αυτές περιγράφονται παρακάτω.

8.4.2.1 Αλλαγή μονάδων μέτρησης

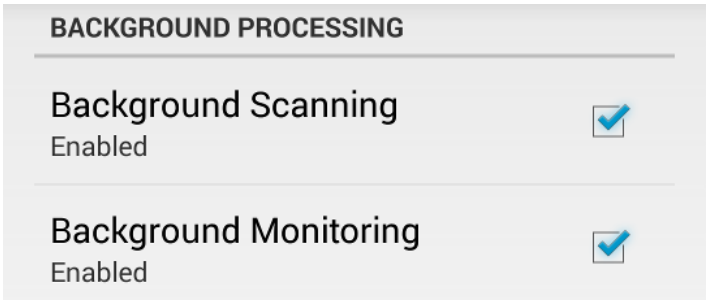
Ο χρήστης μπορεί να ρυθμίσει τις μονάδες μέτρησης που χρησιμοποιούνται στο UI για τη θερμοκρασία και την ατμοσφαιρική πίεση.





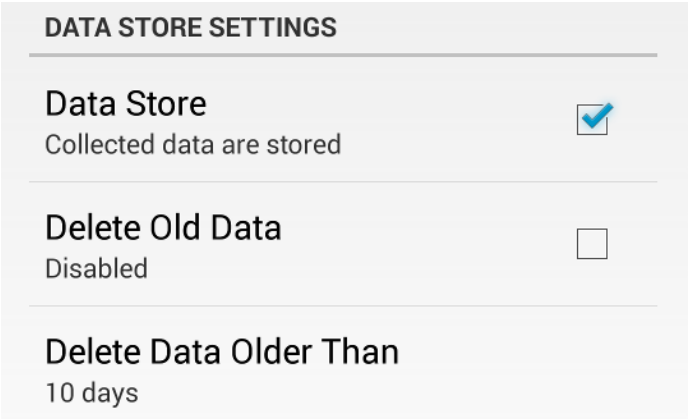
8.4.2.2 Λειτουργία στο παρασκήνιο

Ο χρήστης μπορεί να ενεργοποιήσει ή να απενεργοποιήσει τη λειτουργία των services στο παρασκήνιο. Όταν η λειτουργία στο παρασκήνιο είναι ενεργοποιημένη, οι DeviceScanner και WeatherCollector συνεχίζουν να λειτουργούν και μετά την έξοδο από την εφαρμογή, για αναζήτηση συσκευών, αυτόματη σύνδεση και καταγραφή αποτελεσμάτων.



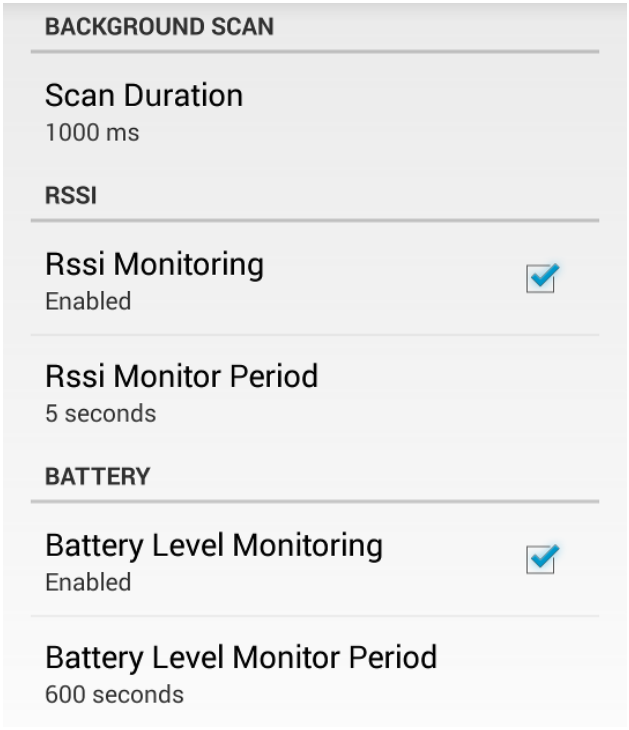
8.4.2.3 Ρυθμίσεις βάσης δεδομένων

Ο χρήστης μπορεί να ενεργοποιήσει ή να απενεργοποιήσει την αποθήκευση των μετρήσεων στη βάση δεδομένων και να ρυθμίσει τη διαγραφή παλαιότερων μετρήσεων.



8.4.2.4 Προχωρημένες ρυθμίσεις

Ο χρήστης μπορεί να ρυθμίσει το χρόνο αναζήτησης του DeviceScanner, καθώς και τις περιοδικές μέτρησεις του επιπέδου της μπαταρίας της απομακρυσμένης συσκευής και της ισχύος του σήματος που λαμβάνεται από αυτή.



8.4.3 Ενημέρωση για αλλαγές

Το αντικείμενο της κλάσης Options κάνει resister ως listener σε όλες τις ρυθμίσεις που ελέγχονται από αυτό (interface SharedPreferences.OnSharedPreferenceChangeListener). Με αυτό τον τρόπο ενημερώνεται για οποιαδήποτε αλλαγή συμβαίνει σε κάποια από τις ρυθμίσεις. Το ίδιο, δίνει τη δυνατότητα σε άλλα στοιχεία της εφαρμογής να ορίσουν callback μεθόδους, με τις οποίες ενημερώνονται για αλλαγές ρυθμίσεων που τα ενδιαφέρουν. Αυτό γίνεται μέσω κατάλληλου interface που ορίζεται από την κλάση Options.

```
public static interface OptionCallback {
    public void backgroundScan (boolean enabled);
    public void backgroundCollect (boolean enabled);
    public void dataStore (boolean enabled);
    public void scanWindow (int win);
    public void rssiMonitor (boolean enabled);
    public void rssiPeriod (int period);
    public void battMonitor (boolean enabled);
    public void battPeriod (int period);
    public void tempUnit (int unit);
    public void pressUnit (int unit);
    public void defDevOpt (int snsr, String key);
    public void deviceName (String devID, String name);
    public void msrmInt (String devID, int snsr, int interval);
    public void snsrMsrmCfg (String devID, int snsr, boolean ntf);
    public void snsrEnabled (String devID, int snsr, boolean enabled);
    public void snsrMasterCfg (String devID, int snsr, boolean msrm,
                                                       boolean ntf);
}
```

Ο WeatherCollector υλοποιεί πολλές από τις παραπάνω μεθόδους. Όταν συμβαίνει μια αλλαγή, αν αυτή αφορά παραμέτρους της συσκευής καταγραφής, ο WeatherCollector στέλνει κατάλληλο μήνυμα στην απομακρυσμένη συσκευή, για την αλλαγή των παραμέτρων. Για παράδειγμα, αν ο χρήστης μεταβάλει τη ρύθμιση ενεργοποίησης ενός αισθητήρα, ο WeatherCollector ενημερώνεται από την Options με κλήση της μεθόδου snsrEnabled. Αν είναι απαραίτητο, ο WeatherCollector στέλνει στη συσκευή καταγραφής την εντολή που προβλέπεται στο προφίλ για τη συγκεκριμένη λειτουργία.

```
...
@Override
public void snsrEnabled(String devID, int snsr, boolean enabled) {
    setSnsrEnabled(devID, snsr, enabled);
}

...

public void setSnsrEnabled (String devID, int snsr, boolean enabled) {
    if (AppDbg.LOG) AppDbg.d(TAG, devID, snsr, "setSnsrEnabled: "+enabled);
    Device dev = findDevice(devID);
    if (dev == null || !dev.connected())
        return;
    dev.setSnsrEnabled(snsr, enabled);
}

...

private class Device {
...
    void setSnsrEnabled (int snsr, boolean enabled) {
        Sensor s = sensors[snsr];
        if (s.enabled != enabled) {
            if (AppDbg.LOG) AppDbg.d(TAG, getID(), snsr,
                                     enabled ? "start" : "stop");
            s.enabled = enabled;
            sendCommand(s, enabled
                           ? Profile.CpCmd.START_SENSOR
                           : Profile.CpCmd.STOP_SENSOR);
        }
    }
...
}
```

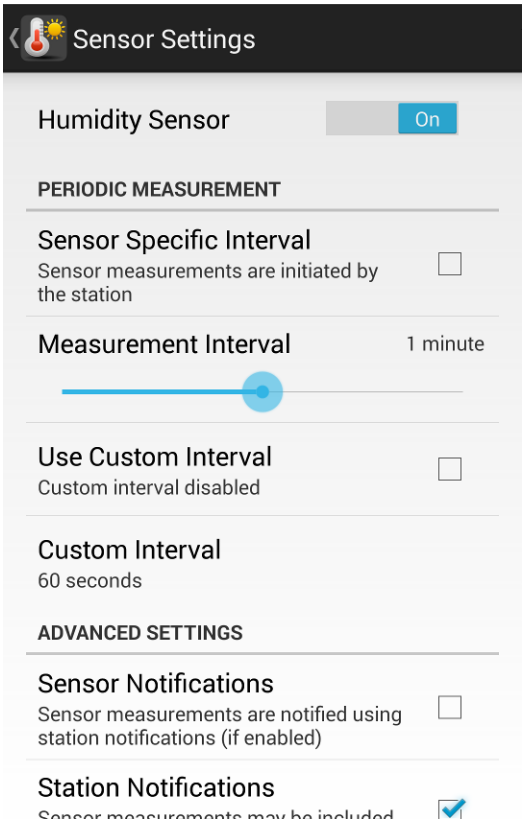
8.4.4 Δημιουργία Οθόνων Ρύθμισης

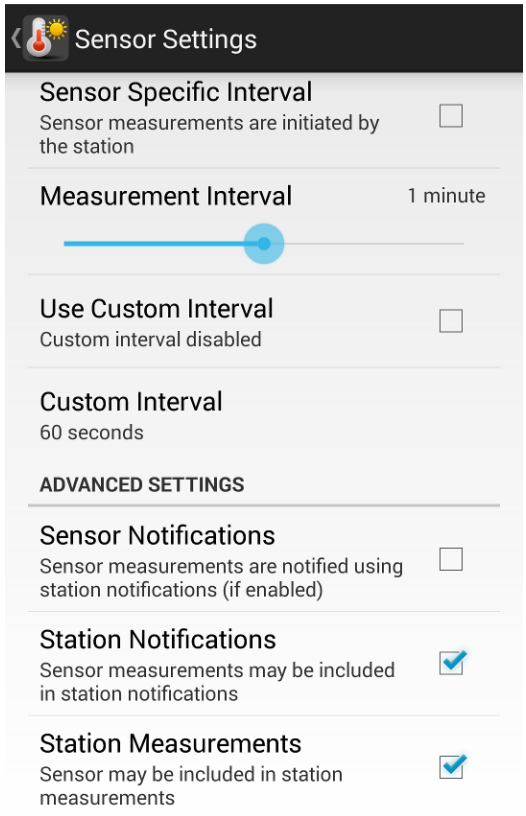
Οι παραπάνω οθόνες ρύθμισης ορίζονται μέσω μιας σειράς XML αρχείων και εμφανίζονται με τη βοήθεια του SettingsActivity, το οποίο φορτώνει τα αρχεία αυτά και τα αντίστοιχα αρχεία ρυθμίσεων, χρησιμοποιώντας το API που παρέχεται από το Android για τον σκοπό αυτό. Ένα πλεονέκτημα της χρήσης του συγκεκριμένου API είναι ότι το UI που παράγεται είναι παρόμοιο με αυτό που χρησιμοποιείται για τις ρυθμίσεις του συστήματος, οπότε ο χρήστης είναι ήδη εξοικειωμένος με αυτό. Επίσης, κάθε αλλαγή που γίνεται, γράφεται αυτόματα στο αντίστοιχο αρχείο ρυθμίσεων, στη ρύθμιση που ορίζεται στο αρχείο XML. Ταυτόχρονα, ενημερώνεται για την αλλαγή η κλάση Options, η οποία, με τη σειρά της, ενημερώνει όλα τα ενδιαφερόμενα στοιχεία της εφαρμογής.

Η κλάση SettingsActivity περιέχει μια σειρά από υποκλάσεις της PreferenceFragment, μία για κάθε οθόνη ρυθμίσεων. Η οθόνη ρύθμισης, που εμφανίζεται στον χρήστη, επιλέγεται από την εφαρμογή κατά τη δημιουργία του SettingsActivity, με την προσθήκη δεδομένων (extra) στο intent δημιουργίας του activity.

Ακολουθεί ένα δείγμα XML αρχείου και η αντίστοιχη οθόνη που προκύπτει.

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:custom="http://schemas.android.com/apk/res-auto" >

<SwitchPreference
    android:key="@string/opt_humid_enabled"
    android:title="@string/humid_snsr"
    android:switchTextOn="@string/switch_text_on"
    android:switchTextOff="@string/switch_text_off"
    android:defaultValue="@string/opt_def_enabled" />

<PreferenceCategory
    android:title="@string/snsr_msrm_int_category_title" >
    <CheckBoxPreference
        android:key="@string/opt_humid_periodicMsrm"
        android:title="@string/snsr_use_sensor_msrm_int_title"
        android:summaryOn="@string/snsr_use_sensor_msrm_int_summary_on"
        android:summaryOff="@string/snsr_use_sensor_msrm_int_summary_off"
        android:defaultValue="@string/opt_def_periodicMsrm" />
    <gr.ntua.microlab.weatherstation.MeasurementIntervalPreference
        android:key="@string/opt_humid_msrmIntIdx"
        android:title="@string/snsr_msrm_int_title"
        android:defaultValue="@string/opt_def_msrmIntIdx" />
    <CheckBoxPreference
        android:key="@string/opt_humid_useCustomMsrmInt"
        android:title="@string/snsr_use_custom_msrm_int_title"
        android:summaryOn="@string/snsr_use_custom_msrm_int_summary_on"
        android:summaryOff="@string/snsr_use_custom_msrm_int_summary_off"
        android:defaultValue="@string/opt_def_useCustomMsrmInt" />
    <gr.ntua.microlab.weatherstation.EditNumberPreference
        android:key="@string/opt_humid_customMsrmInt"
        android:title="@string/snsr_custom_msrm_int_title"
        android:summary="@string/snsr_custom_msrm_int_summary"
        custom:min_value="@string/opt_min_customMsrmInt"
        custom:max_value="@string/opt_max_customMsrmInt"
        custom:show_range="@string/opt_show_range_customMsrmInt"
        custom:unit="@string/seconds"
        custom:unit_one="@string/second"
        android:defaultValue="@string/opt_def_customMsrmInt" />
</PreferenceCategory>

<PreferenceCategory
    android:title="@string/snsr_adv_category_title" >
    <CheckBoxPreference
        android:key="@string/opt_humid_sensorNtf"
        android:title="@string/snsr_ntf_enable_title"
        android:summaryOn="@string/snsr_ntf_enable_summary_on"
        android:summaryOff="@string/snsr_ntf_enable_summary_off"
        android:defaultValue="@string/opt_def_sensorNtf" />
    <CheckBoxPreference
        android:key="@string/opt_humid_masterNtf"
        android:title="@string/snsr_master_ntf_enable_title"
        android:summaryOn="@string/snsr_master_ntf_enable_summary_on"
        android:summaryOff="@string/snsr_master_ntf_enable_summary_off"
        android:defaultValue="@string/opt_def_sensorNtf" />
    <CheckBoxPreference
        android:key="@string/opt_humid_masterMsrm"
        android:title="@string/snsr_master_msrm_enable_title"
        android:summaryOn="@string/snsr_master_msrm_enable_summary_on"
        android:summaryOff="@string/snsr_master_msrm_enable_summary_off"
        android:defaultValue="@string/opt_def_masterMsrm" />
</PreferenceCategory>

</PreferenceScreen>
```





8.5 WeatherDatabase

Για την αποθήκευση των μετρήσεων που στέλνονται από τη συσκευή καταγραφής, δημιουργήσαμε μία βάση δεδομένων. Με αυτό τον τρόπο καθίσταται ευκολότερος ο χειρισμός του ιστορικού μετρήσεων του συστήματος, και ως προς την ανάκτηση, και ως προς τη διαγραφή παλαιότερων δεδομένων. Η δημιουργία της βάσης δεδομένων έγινε με τη βοήθεια της μηχανής SQLite, που περιλαμβάνεται στο Android, και του αντίστοιχου API που παρέχεται για τη χρήση της. Όπως αναφέραμε κατά την παρουσίαση του Android, η SQLite είναι μία μηχανή βάσης δεδομένων, η οποία λειτουργεί μέσα στη διεργασία της εφαρμογής που τη χρησιμοποιεί (δεν έχει δηλαδή εξωτερικό server) και αποθηκεύει όλη τη βάση τοπικά σε ένα μόνο αρχείο. Είναι βελτιστοποιημένη για ενσωματωμένα περιβάλλοντα και, γενικότερα, συσκευές όπως αυτές που τρέχουν το Android. Μπορεί να φανεί ιδιαίτερα χρήσιμη σε περιπτώσεις εφαρμογών όπως η δική μας, στις οποίες απαιτείται απλή χρήση μιας βάσης δεδομένων, αφού παρέχει όλες τις βασικές λειτουργίες που αναμένονται από μια μηχανή βάσης δεδομένων, χωρίς να αυξάνει την πολυπλοκότητα του συστήματος.

Όπως είδαμε, μέσω των ρυθμίσεων της εφαρμογής, δίνεται η δυνατότητα στον χρήστη να απενεργοποιήσει την αποθήκευση των μετρήσεων στη βάση δεδομένων. Επίσης, μπορεί να ρυθμίσει την αυτόματη διαγραφή παλαιότερων από ένα συγκεκριμένο χρονικό όριο δεδομένων, ενώ, μέσω του ιστορικού μετρήσεων, παρέχεται δυνατότητα διαγραφής με επιπλέον ρυθμίσεις, όχι μόνο ως προς την παλαιότητα των δεδομένων, αλλά και ως προς τη συσκευή και το μέγεθος στο οποίο αντιστοιχούν.

Για τη δημιουργία της βάσης δεδομένων, ορίσαμε μια σειρά από tables χρησιμοποιώντας κατάλληλη SQL σύνταξη. Τα δεδομένα που πρέπει να αποθηκευτούν είναι οι τιμές των μεγεθών που λαμβάνονται από τη συσκευή καταγραφής και ο χρόνος που

αντιστοιχεί σε καθεμία από αυτές. Ορίσαμε, λοιπόν, ένα table για κάθε μέγεθος που μας ενδιαφέρει. Επίσης, προκειμένου να υπάρχει σύνδεση μεταξύ των μετρήσεων και των συγκεκριμένων συσκευών από τις οποίες προέρχονται, είναι απαραίτητη η ύπαρξη ενός επιπλέον table, όπου περιέχονται δεδομένα για κάθε γνωστή συσκευή, με την οποία έχει γίνει σύνδεση. Έτσι, κάθε μέτρηση περιέχει επιπλέον μια αναφορά, μέσω foreign key, στη συσκευή από την οποία πραγματοποιήθηκε. Ακολουθεί τμήμα της SQL για τη δημιουργία των tables.

```
create table devices (
   _id integer primary key autoincrement,
   devID text not null,
   name text,
   address text,
   last_conn integer
);

create table msrm (
   _id integer primary key autoincrement,
   device integer,
   msrm_time integer,
   value real,
   foreign key(device) references devices(_id)
);
```

Υπάρχει ένα msrm table για κάθε μέγεθος (π.χ. temp_msrm, humid_msrm). Τα χρονικά δεδομένα αποθηκεύονται ως ακέραιοι. Αντιστοιχούν σε αντικείμενα της κλάσης Date της Java, τα οποία αποθηκεύουν την ημερομηνία σε UTC και μπορούν να αναπαρασταθούν με τη μορφή ακέραιων αριθμών. Το column _id, που χρησιμοποιείται ως primary key, ονομάζεται έτσι, προκειμένου να μπορεί να αναγνωριστεί από στοιχεία του Android API, όπως η λίστα του ιστορικού.

Η βάση δεδομένων δημιουργείται και ελέγχεται από το singleton instance της κλάσης WeatherDatabase. Αυτό χρησιμοποιείται κυρίως από τον collector, για λειτουργίες όπως προσθήκη συσκευών, αν είναι απαραίτητο, κατά τη σύνδεση με αυτές, και μετρήσεων που αυτές στέλνουν, αν η αποθήκευση είναι ενεργοποιημένη. Επίσης, χρησιμοποιείται από το HistoryActivity, ώστε αυτό να παρέχει τα δεδομένα που καθορίζει ο χρήστης, και από το MsrmGraphActivity για τη δημιουργία των αντίστοιχων γραφικών παραστάσεων. Για τον σκοπό αυτό, παρέχει τη δυνατότητα δημιουργίας αντικειμένων τύπου CursorLoader, μιας κλάσης που επιτρέπει την ανάκτηση δεδομένων στο παρασκήνιο, πριν την παρουσίασή τους στον χρήστη, ώστε να μην επιβαρύνεται το UI thread της εφαρμογής.

Η βασική λειτουργία του WeatherDatabase, εκτός από τη δημιουργία της βάσης δεδομένων κατά την πρώτη χρήση της εφαρμογής, είναι η εκτέλεση κατάλληλων SQL queries, προκειμένου να εκτελεστούν συγκεκριμένες διαδικασίες, που απαιτούνται από άλλα στοιχεία της εφαρμογής κατά τη λειτουργία τους. Για τη δημιουργία και εκτέλεση των SQL queries, χρησιμοποιείται το API που παρέχεται από την πλατφόρμα, σε συνδυασμό με παραμέτρους εισόδου που δίνονται από τα υπόλοιπα στοιχεία της εφαρμογής. Ακολουθούν ενδεικτικά SQL queries (οι παράμετροι εισόδου σημειώνονται με <>).

```
-- Εισαγωγή νέας συσκευής (σύνδεση)
insert into devices(devID, address, last_conn)
values (<devID>, <adress>, <time>)


-- Ενημέρωση δεδομένων συσκευής (σύνδεση)
update devices
set address = <address>, last_conn = <time>
where devID = <devID>
```

```
-- Ενημέρωση του χρόνου τελευταίας σύνδεσης (αποσύνδεση)
update devices
set last_conn = <time>
where devID = <devID>


-- Αλλαγή τοπικού ονόματος συσκευής
update devices
set name = <name>
where devID = <devID>


-- Ανάκτηση τοπικού ονόματος συσκευής
select name from devices
where devID = <devID>


-- Εισαγωγή νέας μέτρησης
insert into msrm(device, msrm_time, value)
values (<dev>, <msrm.time>, <msrm.value>)


-- Διαγραφή παλαιών μετρήσεων
-- (από μία ή όλες τις συσκευές)
delete from msrm
where msrm_time < <msrm.time> [and device = <device>]


-- Ανάκτηση τελευταίων μετρήσεων
select * from (
   select * from msrm
   [where device = <dev>]
   order by _id desc limit <last>)
order by _id <desc>


-- Ανάκτηση μετρήσεων με βάση την ημερομηνία
select * from msrm
where [device = <dev> and] msrm_time >= <start> and msrm_time < <end>
order by _id <desc>


-- Στατιστικά συσκευών.
-- Αριθμός μετρήσεων και χρονικό διάστημα στο οποίο έγιναν
select count(msrm_time), min(msrm_time), max(msrm_time)
from msrm
where device = <dev>
```

8.6 BluetoothHelper

Ο BluetoothHelper είναι ένα singleton αντικείμενο, το οποίο χρησιμοποιείται από την εφαρμογή, προκειμένου να πραγματοποιηθούν λειτουργίες σχετικές με το Bluetooth hardware της συσκευής. Δίνει τη δυνατότητα ελέγχου της παρουσίας στη συσκευή, τόσο γενικά του Bluetooth, όσο και ειδικά του BLE hardware, που είναι απαραίτητο για τη λειτουργία της εφαρμογής. Επίσης, ελέγχει αν το Bluetooth είναι ενεργοποιημένο και μπορεί να εκκινήσει τη διαδικασία ενεργοποίησής του, μετά από ερώτηση στον χρήστη. Αποθηκεύει αναφορές στα βασικά αντικείμενα που παρέχονται από το API για το χειρισμό του Bluetooth hardware, τα οποία είναι ο BluetoothManager και ο BluetoothAdapter. Χρησιμοποιείται από όλα τα υπόλοιπα στοιχεία της εφαρμογής που απαιτούν τις υπηρεσίες αυτών των αντικείμενων, δίνοντας τη δυνατότητα ελέγχου, με χρήση απλού κώδικα, της εγκυρότητάς τους πριν από τη χρήση τους.

```
private BluetoothHelper bt;
...
bt = BluetoothHelper.getInstance(this);
bt.addBluetoothStateCallback(this);
...
    if (!bt.ok())
        return;
    bt.getAdapter().startLeScan(this);
...
```

Στον παραπάνω κώδικα, βλέπουμε ότι το αντικείμενο που χρησιμοποιεί τον BluetoothHelper, στην προκειμένη περίπτωση ο DeviceScanner, κάνει register, ώστε να ενημερώνεται από αυτόν, μέσω callback μεθόδων, για συγκεκριμένα γεγονότα που συμβαίνουν. Αυτή είναι μία πολύ σημαντική λειτουργία που παρέχεται από τον BluetoothHelper. Ο BluetoothHelper ορίζει έναν broadcast receiver, τον οποίο κάνει register σε intents τύπου android.bluetooth.adapter.action.STATE_CHANGED. Αυτά στέλνονται από το σύστημα, όταν συμβαίνουν αλλαγές στην κατάσταση του Bluetooth hardware. Οι αλλαγές, για τις οποίες ενδιαφέρεται η εφαρμογή μας, είναι δύο: η ενεργοποίηση του Bluetooth (BluetoothAdapter.STATE_ON) και η έναρξη της διαδικασίας απενεργοποίησής του (BluetoothAdapter.STATE_TURNING_OFF). Ο BluetoothHelper ενημερώνει, με τη σειρά του, όσα στοιχεία της εφαρμογής ενδιαφέρονται για αυτά τα γεγονότα, μέσω ενός interface που ορίζει.

```
public static interface BluetoothStateCallback {
    public void onBluetoothOn ();
    public void onBluetoothTurningOff ();
}
```

Για παράδειγμα, αν ξεκινήσει η απενεργοποίηση του Bluetooth, ο WeatherCollector αποσυνδέεται από τις συσκευές με τις οποίες είναι συνδεμένος, ενώ ο DeviceScanner σταματά το background scanning, αν αυτό είναι ενεργό.

8.7 DeviceScanner

Το DeviceScanner service αναλαμβάνει την αναζήτηση συσκευών BLE, χρησιμοποιώντας το API που παρέχεται από το Android. Η αναζήτηση αυτή μπορεί να γίνεται στο παρασκήνιο, ή μετά από εντολή του χρήστη, όπως, για παράδειγμα, συμβαίνει κατά την αναζήτηση μέσω του DeviceListActivity. Το σύστημα ενημερώνει τον DeviceScanner για κάθε συσκευή BLE, η οποία κάνει advertising, παρέχοντάς του τα advertising και scan response data. Ο DeviceScanner ελέγχει τα δεδομένα αυτά για το weather station service UUID, όπως προβλέπεται από το προφίλ. Αν το UUID περιέχεται στα δεδομένα, τότε θεωρεί ότι η απομακρυσμένη συσκευή είναι weather station και ενημερώνει τα ενδιαφερόμενα στοιχεία της εφαρμογής. Ταυτόχρονα, κάνει broadcast ένα intent, το οποίο περιέχει τη διεύθυνση της συσκευής weather station.

```
private boolean checkScanResponse(final byte[] scanRecord) {
  for (int i = 0; i < scanRecord.length; i += scanRecord[i] + 1) {
    if (scanRecord[i] == 17 && scanRecord[i+1] == 6) {
      ByteBuffer uuid = ByteBuffer.allocate(16)
                                  .order(ByteOrder.LITTLE_ENDIAN)
                                  .put(scanRecord, i+2, 16);
      long lsb = Profile.Service.WEATHER_STATION.getLeastSignificantBits();
      long msb = Profile.Service.WEATHER_STATION.getMostSignificantBits();
      return lsb == uuid.getLong(0) && msb == uuid.getLong(8);
    }
  }
  return false;
}
```

Τα ενδιαφερόμενα στοιχεία της εφαρμογής μπορούν να κάνουν register για ενημέρωση από τον DeviceScanner, χρησιμοποιώντας κατάλληλο interface που παρέχεται από αυτόν.

```
public static interface ScanMonitor {
    public void onDeviceFound (BluetoothDevice device);
    public void onScanComplete ();
    public boolean onStopRequest ();
}
```

Με το παραπάνω interface, εκτός από την ενημέρωση για την εύρεση συσκευών και το τέλος της αναζήτησης, ο DeviceScanner μπορεί να ζητήσει από ενδιαφερόμενα στοιχεία την άδεια να σταματήσει την αναζήτηση στο παρασκήνιο. Αυτό χρησιμεύει, όταν περισσότερα από ένα στοιχεία χρησιμοποιούν τον DeviceScanner. Αν, για παράδειγμα, ένα activity έχει εκκινήσει αναζήτηση στο παρασκήνιο και ο χρήστης το κλείσει, τότε αυτό σταματά την αναζήτηση που είναι σε εξέλιξη. Επειδή η αναζήτηση είναι κοινή για όλα τα στοιχεία, ο DeviceScanner, πριν την τερματίσει, ρωτάει τα υπόλοιπα αν είναι σύμφωνα με αυτό.

Η on-demand αναζήτηση σταματά μετά από ένα, ορισμένο από το στοιχείο που την ξεκίνησε, χρονικό όριο, ή συνεχίζεται μέχρι κάποιο στοιχείο να την τερματίσει. Η αναζήτηση στο παρασκήνιο έχει δύο τύπους. Και στις δύο περιπτώσεις εκκινείται περιοδικά και διαρκεί όσο ορίζεται στις ρυθμίσεις της εφαρμογής. Αυτό, που διαφοροποιεί τους δύο τύπους, είναι η περίοδος με την οποία γίνονται οι αναζητήσεις. Στον πρώτο τύπο, εκτελείται αναζήτηση ανά ένα συγκεκριμένο, σχετικά μεγάλο, χρονικό διάστημα (π.χ. δύο λεπτά). Το διάστημα αυτό πρέπει να είναι τέτοιο ώστε να μην επιβαρύνει τη συσκευή του χρήστη, αλλά, ταυτόχρονα, να επιτρέπει τη γρήγορη εύρεση συσκευών, όταν αυτές είναι σε εμβέλεια. Σε κάθε περίπτωση, ο χρήστης μπορεί να ξεκινήσει on-demand αναζήτηση ή τον δεύτερο τύπο. Ο τελευταίος ξεκινά με πολύ μικρή περίοδο, την οποία αυξάνει σταδιακά, μέχρι να καταλήξει στον πρώτο τύπο. Η χρησιμότητα αυτού του τύπου αναζήτησης είναι ότι επιτρέπει την άμεση εύρεση συσκευών, όταν αυτό είναι περισσότερο πιθανό να συμβεί. Για παράδειγμα, κατά την εκκίνηση της εφαρμογής, θεωρείται ότι μπορεί να υπάρχει μια συσκευή καταγραφής σε εμβέλεια, και χρησιμοποιείται αυτός ο τύπος αναζήτησης. Επίσης, κατά την αποσύνδεση από μια συσκευή, εξαιτίας πτώσης του σήματος, χρησιμοποιείται ο ίδιος τρόπος αναζήτησης, ώστε η απομακρυσμένη συσκευή να ξαναβρεθεί. Έτσι, αν ο χρήστης κινείται στα όρια της εμβέλειας των συσκευών, η επανασύνδεσή τους γίνεται γρήγορα.

8.8 WeatherCollector

Το WeatherCollector service αναλαμβάνει τις περισσότερες από τις λειτουργίες του BLE και του weather station profile, που εκτελούνται από την εφαρμογή. Υλοποιεί, επίσης, την αναμενόμενη συμπεριφορά του ρόλου weather collector, όπως ορίζεται στο weather station profile. Στις λειτουργίες που εκτελεί, περιλαμβάνονται οι ακόλουθες:

- Υλοποιεί όλες τις απαραίτητες λειτουργίες του GATT.
- Συνδέεται με συσκευές weather station που έχουν ανακαλυφθεί από τον DeviceScanner. Η σύνδεση αυτή μπορεί να γίνεται αυτόματα, αν το background collecting είναι ενεργοποιημένο, ή μετά από αίτημα κάποιου activity, το οποίο θέλει να συνδεθεί με μια συγκεκριμένη συσκευή. Προκειμένου να πραγματοποιηθεί η αυτόματη σύνδεση, ορίζεται επίσης ένας broadcast receiver, για την ενημέρωση από τον DeviceScanner. Με τη λήψη του broadcast από τον DeviceScanner, αν το background collecting είναι ενεργοποιημένο, ο receiver εκκινεί το WeatherCollector service, προσθέτοντας στο intent τη διέθυνση της απομακρυσμένης συσκευής. Ο WeatherCollector συνδέεται στη συσκευή αυτή, κατά την εκκίνησή του.
- Διατηρεί μια λίστα με όλες τις συσκευές καταγραφής, με τις οποίες είναι συνδεμένος.
- Αμέσως μετά τη σύνδεση, στέλνει τις απαραίτητες ρυθμίσεις, που είναι αποθηκευμένες τοπικά, στην απομακρυσμένη συσκευή. Επίσης, σε κάθε αλλαγή παραμέτρων λειτουργίας από τον χρήστη, ελέγχει αν είναι απαραίτητο να ενημερώσει και την απομακρυσμένη συσκευή, στέλνοντάς της κατάλληλα μηνύματα. Τα μηνύματα αυτά είναι, είτε εγγραφές στο measurement interval, είτε εντολές που γράφονται στο control point μιας υπηρεσίας.
- Λαμβάνει τα αποτελέσματα των μετρήσεων που στέλνονται από τις συσκευές, τα αποθηκεύει στη βάση δεδομένων και ενημερώνει τα ενδιαφερόμενα στοιχεία. Επίσης, εκτελεί broadcast των αποτελεσμάτων. Με αυτόν τον τρόπο, στοιχεία που ενδιαφέρονται για τα αποτελέσματα, μπορούν να τα λάβουν, χωρίς να χρειάζεται να κάνουν bind με τον WeatherCollector.
- Εκτελεί περιοδικές μετρήσεις της ισχύος του σήματος, που λαμβάνεται από τη συσκευή καταγραφής, και του επιπέδου της μπαταρίας της. Ενημερώνει τα ενδιαφερόμενα στοιχεία.
- Διαγράφει παλαιότερα δεδομένα μετρήσεων από τη βάση δεδομένων, με βάση τις ρυθμίσεις του χρήστη.

Για την ενημέρωση των ενδιαφερόμενων στοιχείων, ο WeatherCollector ορίζει μια σειρά από interfaces, τα οποία μπορούν να χρησιμοποιηθούν από αυτά. Για ενημερώσεις σχετικές με το σήμα και το επίπεδο της μπαταρίας των συνδεμένων συσκευών, χρησιμοποιείται το interface DeviceMonitor.

```
public static interface DeviceMonitor {
    public void onRssi (String devID, int rssi);
    public void onBatteryLevel (String devID, int level);
}
```

Για ενημερώσεις σχετικές με την κατάσταση της σύνδεσης και αποτελέσματα μετρήσεων, χρησιμοποιείται το interface WeatherMonitor. Μέσω αυτού, τα διάφορα activities λαμβάνουν τις μετρήσεις και τις παρουσιάζουν στον χρήστη. Η μέθοδος onDeviceReady καλείται όταν ολοκληρωθεί το service discovery και η ρύθμιση της απομακρυσμένης συσκευής. Η μέθοδος onDisconnectRequest χρησιμοποιείται από τον WeatherCollector, προκειμένου να ζητά την άδεια των registered weather monitors, πριν αποσυνδεθεί από μία συσκευή. Η χρησιμότητα αυτής της λειτουργίας είναι παρόμοια με αυτή που είδαμε στον DeviceScanner. Αν δηλαδή κάποιο στοιχείο της εφαρμογής κλείσει και κάνει

αποσύνδεση από μια συσκευή, ο WeatherCollector ελέγχει αν άλλα στοιχεία χρησιμοποιούν, επίσης, τη συσκευή αυτή.

```
public static interface WeatherMonitor {
    public void onDeviceConnected (String devID, String name);
    public void onDeviceDisconnected (String devID);
    public void onDeviceReady (String devID, String name);
    public boolean onDisconnectRequest (String devID);
    public void onMeasurement (String devID, Profile.Measurement msrm,
                               boolean ntf);
    public void onError (int operation, int status, String devID,
                         BluetoothGatt gatt,
                         BluetoothGattCharacteristic characteristic,
                         BluetoothGattDescriptor descriptor);
}
```

Το τελευταίο interface ενημερώνει για γεγονότα σχετικά με το GATT και σχετίζεται άμεσα με τις αντίστοιχες λειτουργίες του API. Χρησιμοποιείται από το GattViewActivity και μπορεί να απενεργοποιηθεί, αν δεν είναι απαραίτητο.

```
public static interface GattMonitor {
    public void onConnectionStateChange(String devID,
                                        int status, int newState);
    public void onServicesDiscovered(String devID, int status);
    public void onCharacteristicRead(String devID,
                                     BluetoothGattCharacteristic characteristic,
                                     int status);
    public void onCharacteristicWrite(String devID,
                                     BluetoothGattCharacteristic characteristic,
                                     int status);
    public void onCharacteristicChanged(String devID,
                                     BluetoothGattCharacteristic characteristic);
    public void onDescriptorRead(String devID,
                                     BluetoothGattDescriptor descriptor,
                                     int status);
    public void onDescriptorWrite(String devID,
                                     BluetoothGattDescriptor descriptor,
                                     int status);
    public void onReadRemoteRssi(String devID, int rssi, int status);
    public void onReliableWriteCompleted(String devID, int status);
}
```

Το προφίλ ορίζει ότι δεν επιτρέπεται στον collector να στείλει δεύτερη εντολή στον weather station (εκτός της “Update Now”), πριν λάβει από τον τελευταίο απάντηση, μέσω GATT indication, για την προηγούμενη εντολή που έστειλε. Προκειμένου να υλοποιηθεί αυτή η συμπεριφορά, ο WeatherCollector περιλαμβάνει μια ουρά εντολών προς εκτέλεση ανά συσκευή, οι οποίες στέλνονται ακολουθιακά στον weather station.

Κατά την περιγραφή του BLE API του Android, αναφέραμε κάποιους περιορισμούς που αυτό παρουσιάζει. Ένας από αυτούς είναι ο περιορισμένος αριθμός notifications ή indications, τα οποία μπορούν να είναι ενεργοποιημένα τοπικά, ταυτόχρονα. Ο αριθμός αυτός ήταν 4 στο API level 18 και έγινε 7 στο API level 19. Αν υπάρχουν πολλοί αισθητήρες, τότε δεν είναι δυνατόν να έχουν όλοι ενεργοποιημένες, ταυτόχρονα, τις αποστολές μετρήσεων. Αυτό μπορεί να αντιμετωπιστεί με τη χρήση του master service measurement, το οποίο απαιτεί την ενεργοποίηση μόνο των δικών του αποστολών. Επιπλέον, πρέπει ο collector να λαμβάνει τα indications από τον weather station, μετά από εντολές που στέλνει σε αυτόν. Για να γίνει η λήψη των indications πρέπει να ενεργοποιηθούν και τοπικά. Η τεχνική που ακολουθήσαμε στην εφαρμογή μας ήταν να ενεργοποιούμε τα τοπικά indications, μόνο για το control point από το οποίο αναμένουμε να λάβουμε απάντηση. Αυτό είναι εφικτό επειδή, λόγω του flow control που ορίζει το προφίλ για την αποστολή εντολών, γνωρίζουμε σε κάθε περίπτωση σε ποιο control point θα γίνει indication από τον weather station, το οποίο,

φυσικά, είναι αυτό στο οποίο στάλθηκε η εντολή. Μετά τη λήψη του indication, τα τοπικά indications για το control point απενεργοποιούνται.

Ένας δεύτερος περιορισμός είναι η έλλειψη κάποιου είδους ουράς, στο ίδιο το BLE API, για την ακολουθιακή εκτέλεση λειτουργιών του GATT που ορίζουν transaction, και, ως εκ τούτου, δεν μπορούν να εκτελούνται ταυτόχρονα. Αυτές οι λειτουργίες, από την πλευρά του WeatherCollector, είναι τα Read και Write σε characteristics και descriptors. Για την αντιμετώπισή του περιορισμού αυτού, ο WeatherCollector περιλαμβάνει μια ουρά GATT λειτουργιών προς εκτέλεση ανά συσκευή, ώστε τα transactions να εκτελούνται ακολουθιακά.

8.9 User Interface

Ακολουθεί περιγραφή των βασικών activities που περιλαμβάνονται στην εφαρμογή, και τα οποία αποτελούν, μαζί με τις οθόνες ρυθμίσεων, το user interface του συστήματος καταγραφής καιρού.

8.9.1 DeviceActivityBase

Η κλάση DeviceActivityBase χρησιμοποιείται ως βασική κλάση για τη δημιουργία activities, τα οποία πραγματοποιούν λειτουργίες όπως αναζήτηση και σύνδεση με συσκευές καταγραφής, καθώς και λήψη μετρήσεων από αυτές. Η DeviceActivityBase υλοποιεί τις γενικές λειτουργίες ενός τέτοιου activity και αυτοματοποιεί πολλές από τις διαδικασίες που απαιτούνται. Περιλαμβάνει λειτουργίες όπως:

- Σύνδεση (bind) με τον DeviceScanner και register ενός ScanMonitor interface.
- Σύνδεση (bind) με τον WeatherCollector και register ενός DeviceMonitor interface και ενός WeatherMonitor interface.
- Αυτόματη έναρξη αναζήτησης συσκευών καταγραφής.
- Αυτόματη σύνδεση με συσκευές καταγραφής.
- Χρήση του BluetoothHelper για τον έλεγχο της κατάστασης του Bluetooth.
- Έλεγχος παρουσίας στη συσκευή του απαραίτητου BLE hardware, με δυνατότητα τερματισμού του activity σε περίπτωση αποτυχίας.
- Έλεγχος αν το Bluetooth είναι ενεργοποιημένο, με δυνατότητα ενεργοποίησής του, μετά από ερώτηση στον χρήστη, και τερματισμού του activity, σε περίπτωση μη ενεργοποίησης.
- Δημιουργία στοιχείων, στο μενού του activity, για λειτουργίες όπως σύνδεση, αποσύνδεση, έναρξη και τερματισμός αναζήτησης.

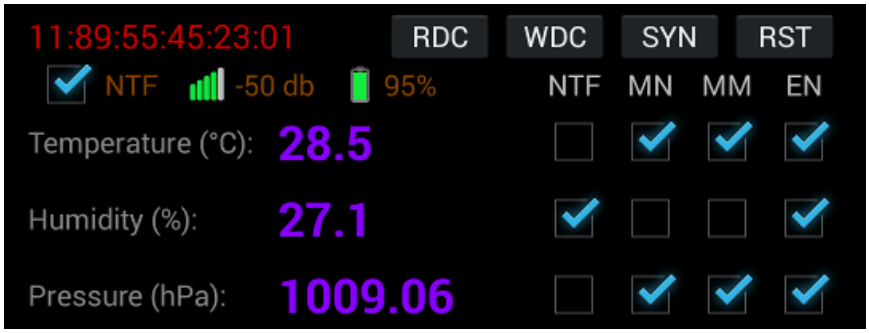
Η DeviceActivityBase περιλαμβάνει μια σειρά από μεθόδους που μπορούν να γίνουν override από τις υποκλάσεις της. Χρησιμοποιούνται για την ενημέρωση των υποκλάσεων για γεγονότα όπως σύνδεση, αποσύνδεση και λήψη αποτελεσμάτων, καθώς και για τον ορισμό της συμπεριφοράς της DeviceActivityBase (π.χ. τερματισμός activity, στις περιπτώσεις που αναφέρθηκαν παραπάνω). Η κλάση DeviceActivityBase υλοποιήθηκε με το σκεπτικό της υποστήριξης μόνο μιας σύνδεσης με μια συσκευή καταγραφής, που είναι και η πιο συνηθισμένη περίπτωση. Όμως, είναι δυνατόν να χρησιμοποιηθεί, με κατάλληλη παραμετροποίηση, και για την υποστήριξη πολλαπλών συσκευών, όπως συμβαίνει για παράδειγμα στην περίπτωση του DeviceListActivity. Γενικότερα, η λειτουργία της DeviceActivityBase μπορεί να παραμετροποιηθεί πλήρως από τις υποκλάσεις της, με απενεργοποίηση λειτουργιών που δεν απαιτούνται, και αλλαγή της συμπεριφοράς της με override κατάλληλων μεθόδων.

Κατά την υλοποίηση της εφαρμογής, ορίσαμε τέσσερις υποκλάσεις της DeviceActivityBase, τα activities WeatherLogActivity, GattViewActivity, DeviceListActivity και CollectorActivity.

8.9.2 WeatherLogActivity

Το WeatherLogActivity δημιουργεί ένα βασικό UI, όπου παρουσιάζονται τα αποτελέσματα μετρήσεων που λαμβάνονται από τη συσκευή καταγραφής, καθώς και το debug log της εφαρμογής. Επίσης, δίνεται η δυνατότητα άμεσης ρύθμισης παραμέτρων λειτουργίας, απευθείας από την αρχική οθόνη, καθώς και εκτέλεσης μετρήσεων.

Με το πάτημα στην τιμή κάποιου μεγέθους, εκτελείται άμεση μέτρηση στο αντίστοιχο μέγεθος, ενώ με πάτημα στη διεύθυνση της συσκευής, εκτελείται άμεση μέτρηση σε όλα τα μεγέθη. Αυτό γίνεται με αποστολή της εντολής "Update Now" στη συσκευή καταγραφής, στην πρώτη περίπτωση στο sensor service που αντιστοιχεί στο μέγεθος, και στη δεύτερη περίπτωση στο master service. Με παρατεταμένο πάτημα στην τιμή κάποιου μεγέθους, διαβάζεται από τη συσκευή καταγραφής η τελευταία μέτρηση που έγινε από τον αντίστοιχο αισθητήρα.

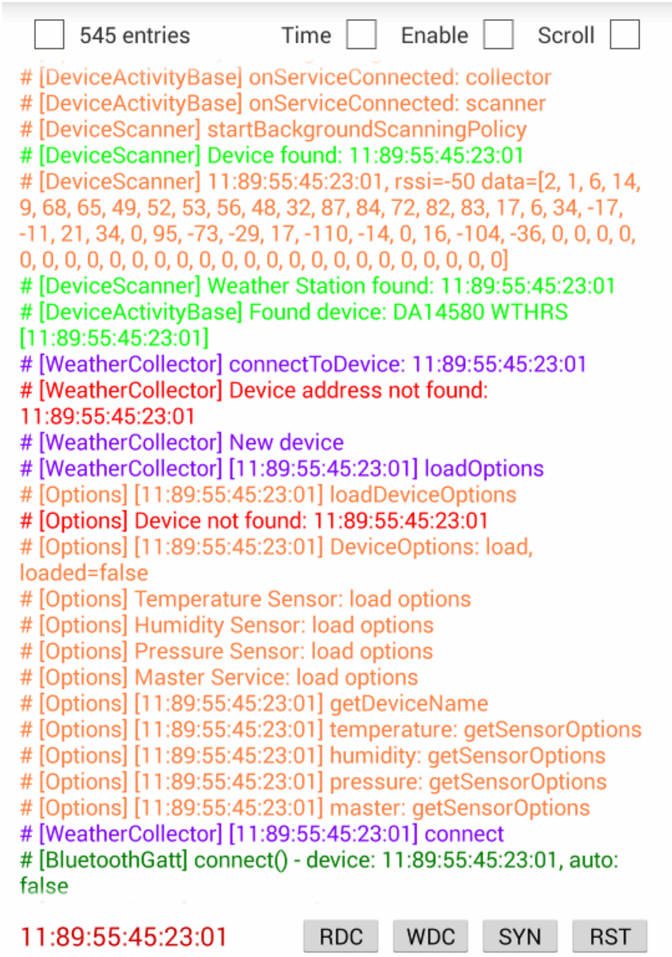


Οι ρυθμίσεις και οι ενέργειες που μπορούν να γίνουν απευθείας από την οθόνη του WeatherLogActivity, παρουσιάζονται στον παρακάτω πίνακα.

NTF	Ενεργοποίηση αποστολών μετρήσεων (notifications).
MN	Δυνατότητα αποστολής μέσω του master service measurement.
MM	Ενεργοποίηση μετρήσεων από το master service.
EN	Ενεργοποίηση αισθητήρα.
RDC	Διάβασμα παραμέτρων λειτουργίας από τη συσκευή καταγραφής και αποθήκευση τοπικά.
WDC	Γράψιμο των παραμέτρων λειτουργίας, που είναι αποθηκευμένες τοπικά, στη συσκευή καταγραφής. Η λειτουργία αυτή είναι παρόμοια με εκείνη, που εκτελείται από τον WeatherCollector μετά τη σύνδεση.
SYN	Αποστολή εντολής συγχρονισμού μετρήσεων.
RST	Αποστολή εντολής επαναφοράς (reset) όλων των αισθητήρων.

Ακολουθούν μερικά παραδείγματα debug log.

Στην παρακάτω εικόνα, παρουσιάζεται η εύρεση μιας συσκευής καταγραφής και η έναρξη της διαδικασίας σύνδεσης. Αρχικά το WeatherLogActivity, μέσω της υπερκλάσης του (DeviceActivityBase), κάνει bind με τα δύο services, WeatherCollector και DeviceScanner. Το ακόλουθο log ξεκινά με την ολοκλήρωση του bind. Μετά τη σύνδεση με τον DeviceScanner, το activity εκκινεί την αναζήτηση συσκευών. Βρίσκεται άμεσα μια BLE συσκευή και ο DeviceScanner ελέγχει τα advertising και scan response data για το weather station UUID. Με την εύρεση του τελευταίου, ενημερώνεται το activity, το οποίο ξεκινά τη διαδικασία σύνδεσης, καλώντας μια μέθοδο στον WeatherCollector. Ο τελευταίος ελέγχει αν η συσκευή είναι ήδη στη λίστα που διατηρεί, και, αν είναι απαραίτητο, την προσθέτει. Στη συνέχεια, φορτώνει μέσω της Options τις ρυθμίσεις της συγκεκριμένης συσκευής. Τέλος, εκκινεί τη διαδικασία σύνδεσης, καλώντας τη μέθοδο που παρέχεται από το Android API.



Στην εικόνα της διπλανής σελίδας, παρουσιάζεται μέρος της διαδικασίας service discovery που εκτελείται μετά τη σύνδεση. Το log αυτό παράγεται από το BLE stack του Android. Αρχικά, παρατηρούμε την ανακάλυψη των primary services του weather station, ενώ αμέσως μετά ξεκινά η ανακάλυψη των χαρακτηριστικών τους.

```
# [BluetoothGatt] onGetService() - Device=11:89:55:45:23:01 UUID=00001800-0000-1000-8000-00805f9b34fb
# [BluetoothGatt] onGetService() - Device=11:89:55:45:23:01 UUID=00001801-0000-1000-8000-00805f9b34fb
# [BluetoothGatt] onGetService() - Device=11:89:55:45:23:01 UUID=0000180a-0000-1000-8000-00805f9b34fb
# [BluetoothGatt] onGetService() - Device=11:89:55:45:23:01 UUID=0000180f-0000-1000-8000-00805f9b34fb
# [BluetoothGatt] onGetService() - Device=11:89:55:45:23:01 UUID=dc981000-f292-11e3-b75f-002215f5ef22
# [BluetoothGatt] onGetService() - Device=11:89:55:45:23:01 UUID=dc981100-f292-11e3-b75f-002215f5ef22
# [BluetoothGatt] onGetService() - Device=11:89:55:45:23:01 UUID=dc981200-f292-11e3-b75f-002215f5ef22
# [BluetoothGatt] onGetService() - Device=11:89:55:45:23:01 UUID=dc981300-f292-11e3-b75f-002215f5ef22
# [BluetoothGatt] onGetCharacteristic() - Device=11:89:55:45:23:01
UUID=00002a00-0000-1000-8000-00805f9b34fb
# [BluetoothGatt] onGetCharacteristic() - Device=11:89:55:45:23:01
UUID=00002a01-0000-1000-8000-00805f9b34fb
# [BluetoothGatt] onGetCharacteristic() - Device=11:89:55:45:23:01
UUID=00002a02-0000-1000-8000-00805f9b34fb
# [BluetoothGatt] onGetCharacteristic() - Device=11:89:55:45:23:01
UUID=00002a04-0000-1000-8000-00805f9b34fb
```

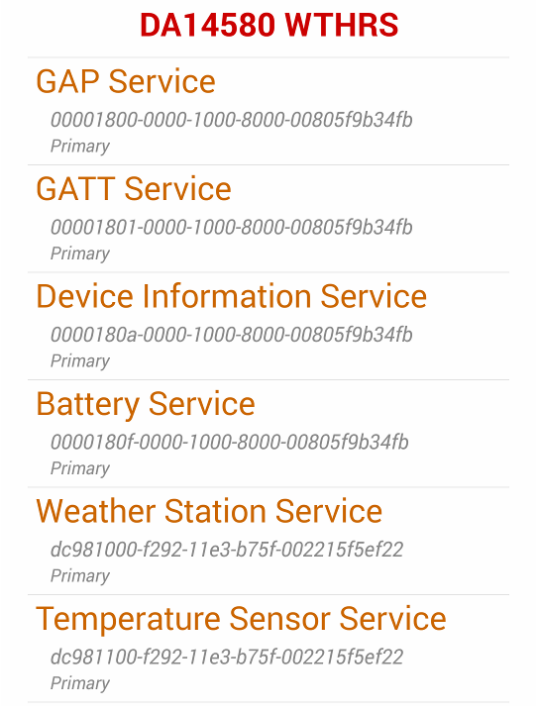
Στην παρακάτω εικόνα, παρουσιάζεται η διαδικασία της λήψης αποτελεσμάτων από τη συσκευή καταγραφής, μετά από εντολή "Update Now" στο master service. Στην προκειμένη περίπτωση έχουμε τρεις ενεργοποιημένους αισθητήρες (θερμοκρασία, πίεση, υγρασία). Έχουμε ενεργοποιήσει τις αποστολές του humidity sensor service, οπότε η μέτρηση της υγρασίας στέλνεται με ξεχωριστό notification. Αρχικά, βλέπουμε την επιτυχία της εγγραφής στο control point του master service. Η συσκευή καταγραφής εκκινεί μετρήσεις στους ενεργοποιημένους αισθητήρες και στέλνει τα αποτελέσματα. Το πρώτο notification που λαμβάνεται είναι για το master service measurement, το οποίο περιέχει τα μεγέθη της θερμοκρασίας και της πίεσης. Στη συνέχεια, λαμβάνεται το notification για το humidity sensor measurement. Οι μετρήσεις στέλνονται και στη βάση δεδομένων. Οι παραπάνω διαδικασίες εκτελούνται από τον WeatherCollector. Ο τελευταίος ενημερώνει το activity για τη λήψη των αποτελεσμάτων, ενώ, επίσης, κάνει broadcast τα αποτελέσματα. Το broadcast λαμβάνεται από το WeatherStationWidget, το οποίο ανανεώνει τα περιεχόμενά του.

```
# [WeatherCollector] [11:89:55:45:23:01] master: control point
written successfully
# [BluetoothGatt] onNotify() - Device=11:89:55:45:23:01
UUID=dc981001-f292-11e3-b75f-002215f5ef22
# [GATT] [11:89:55:45:23:01] onCharacteristicChanged, Weather
Station Measurement
# [WeatherCollector] [11:89:55:45:23:01] master: measurement
notification
# [WeatherCollector] Master Service: measurement
dev=11:89:55:45:23:01
# [WeatherCollector] Measurement: Mon Jul 14 11:57:24 EEST
2014
# [WeatherCollector] Temperature: 28.700001
# [WeatherCollector] Pressure: 101101.000000
# [WeatherDatabase] addMeasurement: 11:89:55:45:23:01
# [BluetoothGatt] onNotify() - Device=11:89:55:45:23:01
UUID=dc981201-f292-11e3-b75f-002215f5ef22
# [GATT] [11:89:55:45:23:01] onCharacteristicChanged,
Humidity Measurement
# [WeatherCollector] [11:89:55:45:23:01] humidity:
measurement notification
# [WeatherCollector] Humidity Sensor: measurement
dev=11:89:55:45:23:01
# [WeatherCollector] Measurement: Mon Jul 14 11:57:24 EEST
2014
# [WeatherCollector] Humidity: 23.299999
# [WeatherLogActivity] onMeasurement
# [WeatherDatabase] addMeasurement: 11:89:55:45:23:01
# [WeatherStationWidget] onReceive
# [Options] getInstance:
android.app.ReceiverRestrictedContext@4289df18
# [WeatherLogActivity] onMeasurement
```

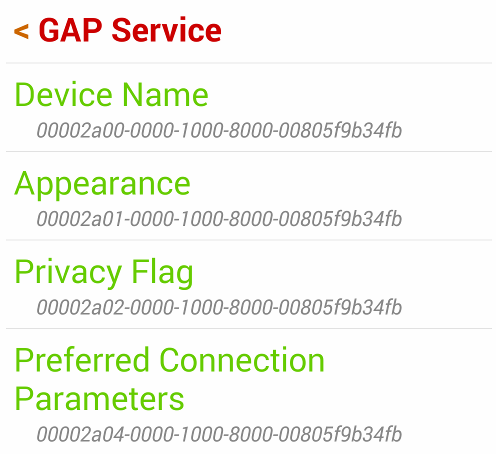
8.9.3 GattViewActivity

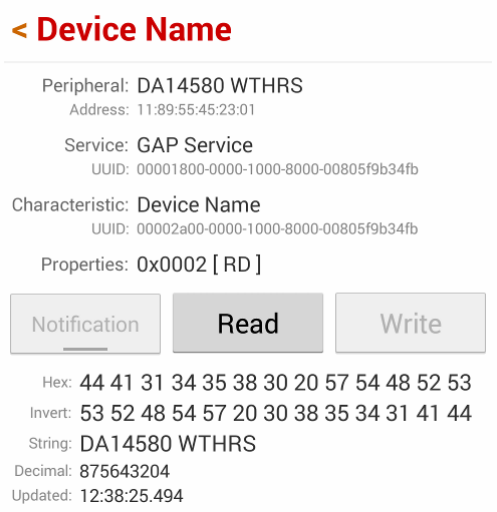
Το GattViewActivity είναι βασισμένο σε κώδικα που παρέχεται από το Bluetooth SIG (Application Accelerator v1.1 – BLE Demo). Χρησιμοποιώντας το interface GattMonitor του WeatherCollector, μπορεί να ανακτά τη λίστα των υπηρεσιών από μια απομακρυσμένη συσκευή, τις οποίες και παρουσιάζει στον χρήστη. Αναγνωρίζει όλες τις υπηρεσίες που ορίζονται στο weather station profile και τα αντίστοιχα χαρακτηριστικά. Δίνει τη δυνατότητα ανάκτησης και εγγραφής τιμών χαρακτηριστικών και client characteristic configuration descriptors. Χρησιμοποιήθηκε για τον έλεγχο της σωστής λειτουργίας του συστήματος καταγραφής καιρού.

Στην παρακάτω εικόνα βλέπουμε μέρος της λίστας με τις υπηρεσίες που περιλαμβάνονται στη συσκευή καταγραφής.

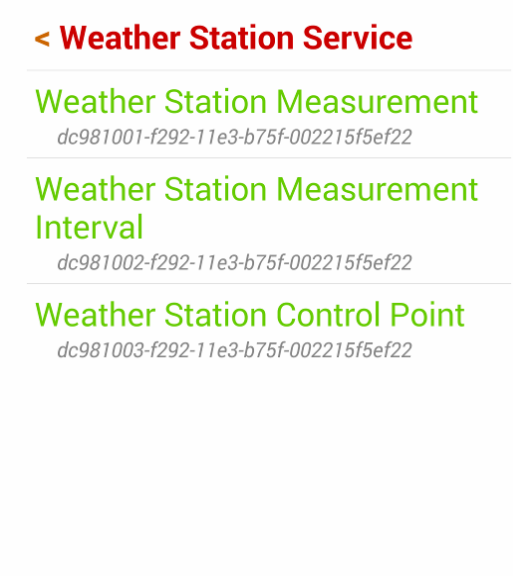


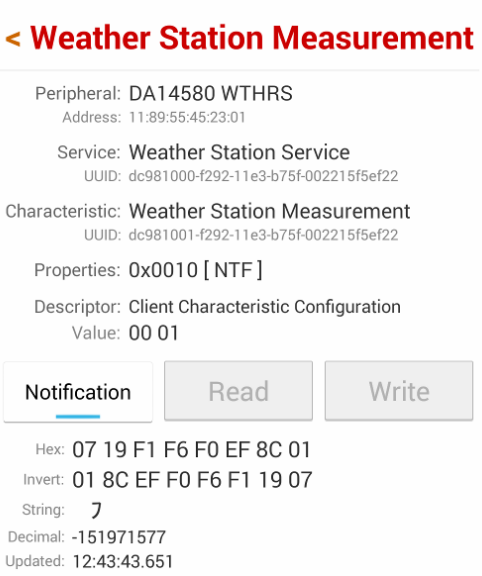
Στις ακόλουθες εικόνες, βλέπουμε τα χαρακτηριστικά του GAP service της συσκευής και το χαρακτηριστικό Device Name.





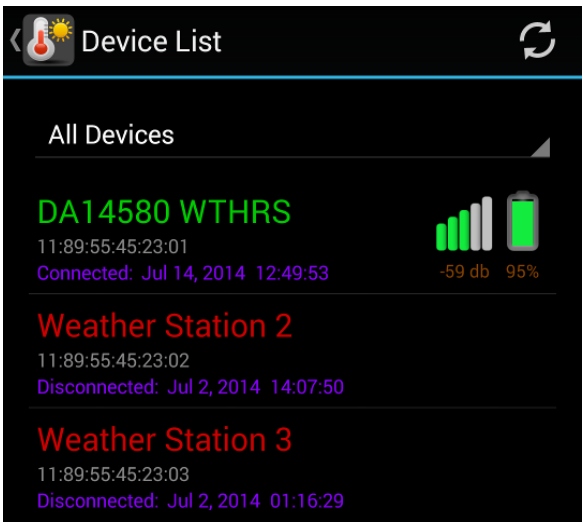
Στις ακόλουθες εικόνες, βλέπουμε τα χαρακτηριστικά του weather station service της συσκευής και το χαρακτηριστικό weather station measurement. Τα δεδομένα που ελήφθησαν με το τελευταίο notification είναι διαθέσιμα τοπικά και παρουσιάζονται στην οθόνη.



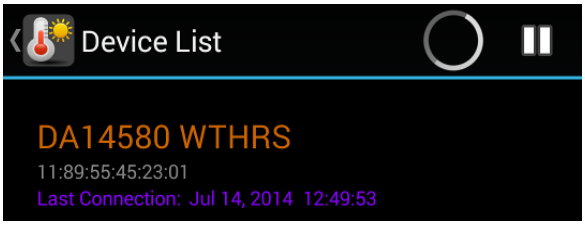


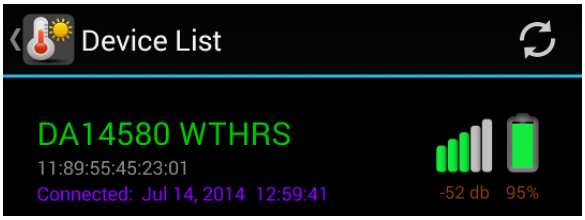
8.9.4 DeviceListActivity

Το DeviceListActivity παρουσιάζει στον χρήστη μια λίστα με τις γνωστές συσκευές, συνδεμένες ή αποσυνδεμένες. Επίσης, μπορεί να χρησιμοποιηθεί για την εκτέλεση αναζήτησης, οπότε παρουσιάζει και τις συσκευές που βρέθηκαν. Σε αυτή την περίπτωση, ο χρήστης μπορεί να συνδεθεί με μια συσκευή πατώντας πάνω στο αντίστοιχο στοιχείο της λίστας. Επίσης δίνεται η δυνατότητα αλλαγής των ρυθμίσεων κάθε συσκευής με παρατεταμένο πάτημα (long click) του αντίστοιχου στοιχείου. Αν η συσκευή είναι συνδεμένη, τότε οι αλλαγές στέλνονται άμεσα, διαφορετικά στέλνονται από τον WeatherCollector μετά τη σύνδεση.



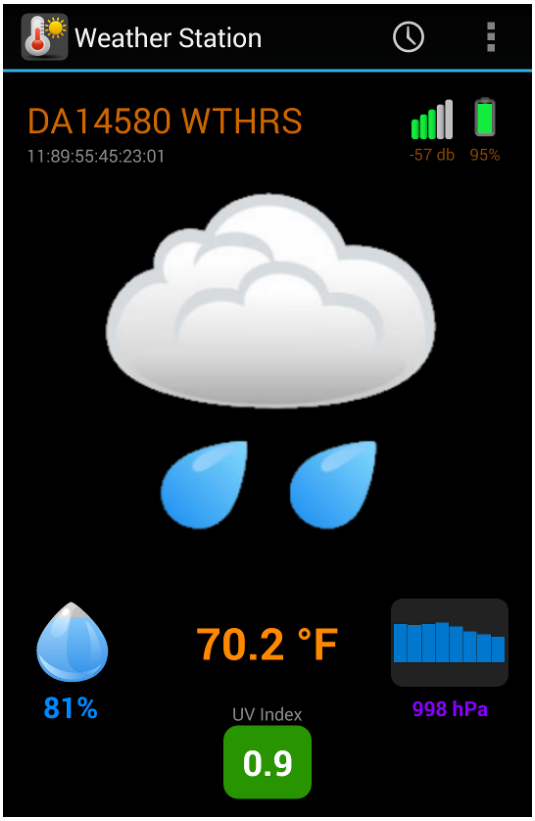
Στις ακόλουθες εικόνες βλέπουμε τη διαδικασία της αναζήτησης συσκευών και την ακόλουθη σύνδεση.





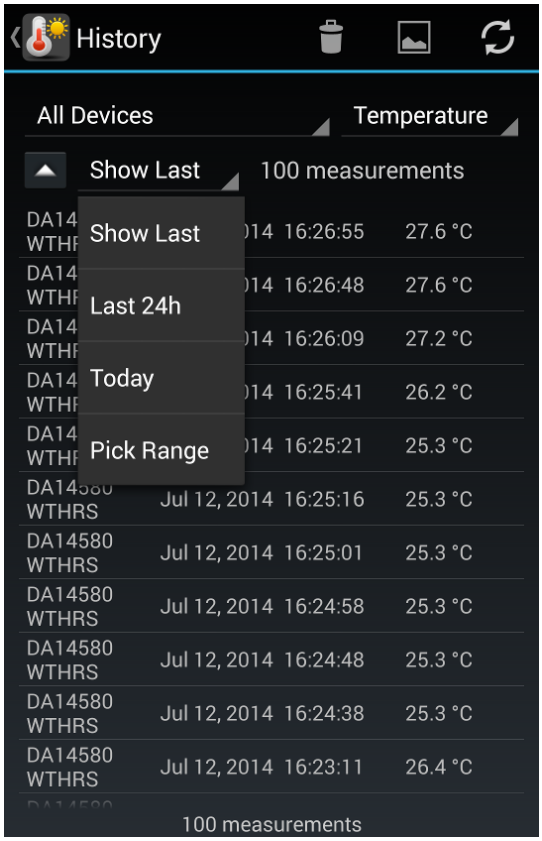
8.9.5 CollectorActivity

Το CollectorActivity είναι το αρχικό activity της εφαρμογής. Δημιουργεί ένα UI μέσω του οποίου παρουσιάζει στον χρήστη τα αποτελέσματα των μετρήσεων. Επίσης, παρέχει πρόσβαση σε όλα τα άλλα activities της εφαρμογής και στις οθόνες ρυθμίσεων.



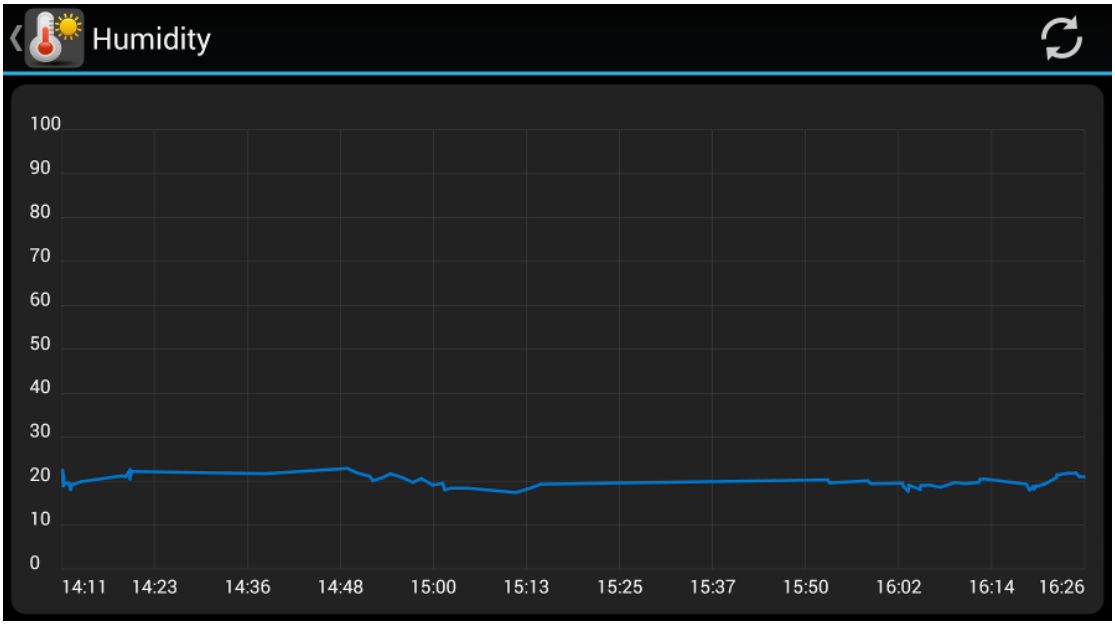
8.9.6 HistoryActivity

Το HistoryActivity δίνει τη δυνατότητα στον χρήστη να δει το ιστορικό των μετρήσεων, που είναι αποθηκευμένες στη βάση δεδομένων. Ο χρήστης μπορεί να παραμετροποιήσει τα αποτελέσματα που εμφανίζονται, ορίζοντας το μέγεθος και τη συσκευή που τον ενδιαφέρει, καθώς και τις ημερομηνίες λήψης των μετρήσεων. Στην περίπτωση που παρουσιάζονται οι τελευταίες μετρήσεις, η λίστα ανανεώνεται με κάθε νέα μέτρηση. Το ιστορικό δίνει επιπλέον τη δυνατότητα διαγραφής δεδομένων από τη βάση, με μεγαλύτερο έλεγχο από αυτόν που παρέχεται από την αντίστοιχη λειτουργία του WeatherCollector. Τέλος, υπάρχει η δυνατότητα δημιουργίας γραφικής παράστασης με τα αποτελέσματα, με τη βοήθεια του MsrmGraphActivity.



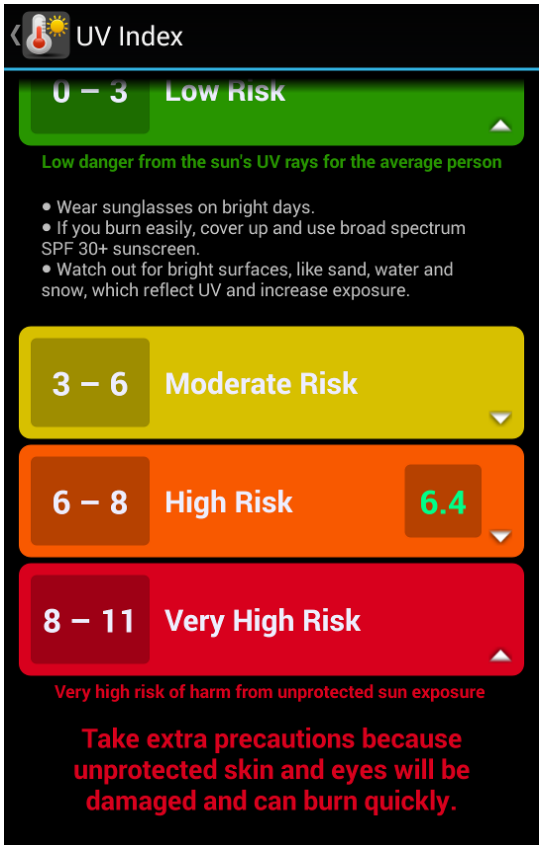
8.9.7 MsrmGraphActivity

Το MsrmGraphActivity παρουσιάζει στον χρήστη μια γραφική παράσταση με αποτελέσματα μετρήσεων ενός μεγέθους. Μέσω κατάλληλων gestures, δίνεται η δυνατότητα αλλαγής του τμήματος της γραφικής παράστασης που φαίνεται στην οθόνη. Για τη δημιουργία των γραφικών παραστάσεων χρησιμοποιήθηκε η βιβλιοθήκη GraphView (v3.1.2).



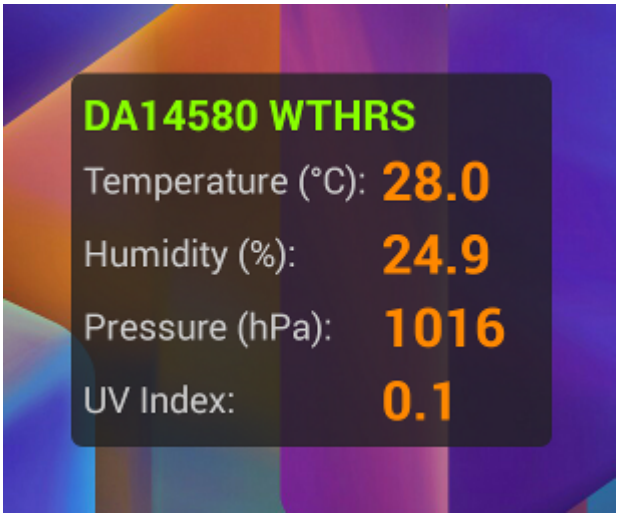
8.9.8 UVIndexActivity

Το UVIndexActivity παρουσιάζει στον χρήστη τα απαραίτητα μέτρα που πρέπει να λάβει, για την προστασία του από την υπεριώδη ακτινοβολία του ήλιου, με βάση την τρέχουσα τιμή του UVIndex. Κατά τη δημιουργία του UVIndexActivity, η κατηγορία, στην οποία αντιστοιχεί η τρέχουσα τιμή του UVIndex, επιλέγεται αυτόματα. Ο χρήστης μπορεί στη συνέχεια να ελέγξει και τις άλλες κατηγορίες.



8.9.9 WeatherStationWidget

Το WeatherStationWidget μπορεί να προστεθεί στη home screen της συσκευής του χρήστη και, χρησιμοποιώντας τις υπηρεσίες του WeatherCollector, να εμφανίζει αποτελέσματα μετρήσεων σε αυτήν, με αυτόματη ανανέωση.



Συντομογραφίες

ADC	Analog to Digital Converter
ADT	Android Development Tools
AFH	Adaptive Frequency Hopping
AMP	Alternate MAC PHY
API	Application Programming Interface
ATT	Attribute Protocol
BER	Bit Error Rate
BLE	Bluetooth Low Energy
BR	Basic Rate
CID	Channel Identifier
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSRK	Connection Signature Resolving Key
EDR	Enhanced Data Rate
FSK	Frequency Shift Keying
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GFSK	Gaussian Frequency Shift Keying
GPIO	General Purpose Input Output
HCI	Host Controller Interface
IDE	Integrated Development Environment
IO	Input/Output
IRK	Identity Resolving Key
ISM	Industrial Scientific Medical
L2CAP	Logical Link Control and Adaptation Protocol
LE	Low Energy
LL	Link Layer
LLM	Link Layer Manager
LSB	Least Significant Bit
LTK	Long Term Key
MAC	Media Access Control
MIC	Message Integrity Check
MITM	Man In The Middle
MSB	Most Significant Bit
MTU	Maximum Transmission Unit
NVDS	Non Volatile Data Storage

OTP	One Time Programmable
PDU	Protocol Data Unit
PHY	Physical Layer
PSK	Phase Shift Keying
RAM	Random Access Memory
ROM	Read Only Memory
RSSI	Received Signal Strength Indication
RTOS	Real Time Operating System
SDK	Software Development Kit
SIG	Special Interest Group
SMP	Security Manager Protocol
SoC	System on a Chip
SSP	Secure Simple Pairing
STK	Short Term Key
SWD	Serial Wire Debug
TK	Temporary Key
UI	User Interface
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier

Βιβλιογραφία

Bluetooth

Βιβλία

1. Robin Heydon, "Bluetooth Low Energy – The Developer's Handbook", Prentice Hall, 2012
2. Kevin Townsend, Carles Cufí, Akiba, Robert Davidson, "Getting Started with Bluetooth Low Energy", O'Reilly, 2014

Specifications

1. "Bluetooth Core Specification Version 4.0", Bluetooth SIG, 2010
2. "Bluetooth Core Specification Version 4.1", Bluetooth SIG, 2013
3. "Bluetooth Core Specification Supplement (CSS) v4", Bluetooth SIG, 2013
4. "BATTERY SERVICE SPECIFICATION, V10r00", Bluetooth SIG
5. "DEVICE INFORMATION SERVICE, V11r00"
6. "PROXIMITY PROFILE, v10r00", Bluetooth SIG
7. "IMMEDIATE ALERT SERVICE, v10r00", Bluetooth SIG
8. "LINK LOSS SERVICE, v10r00", Bluetooth SIG
9. "TX POWER SERVICE, v10r00", Bluetooth SIG
10. "Find Me Profile, V10r00", Bluetooth SIG
11. "GLUCOSE PROFILE, V10", Bluetooth SIG
12. "GLUCOSE SERVICE, V10", Bluetooth SIG
13. "BLOOD PRESSURE PROFILE, V10r00", Bluetooth SIG
14. "BLOOD PRESSURE SERVICE, V10r00", Bluetooth SIG
15. "HEART RATE PROFILE, v10r00", Bluetooth SIG
16. "HEART RATE SERVICE, v10r00", Bluetooth SIG
17. "HEALTH THERMOMETER PROFILE, v10r00", Bluetooth SIG
18. "HEALTH THERMOMETER SERVICE, v10r00", Bluetooth SIG
19. "LOCATION AND NAVIGATION PROFILE, V10", Bluetooth SIG
20. "LOCATION AND NAVIGATION SERVICE, V10", Bluetooth SIG
21. "TIME PROFILE, v10r00", Bluetooth SIG
22. "Developing a Bluetooth Product – Getting Started Guide", Bluetooth SIG
23. "Using the Bluetooth Brand – Getting Started Guide", Bluetooth SIG
24. "Personal Health Devices Transcoding White Paper, V12r00", Bluetooth SIG
25. "BLE 101 – Bluetooth Low Energy", Bluetooth SIG presentation, Bluetooth Smart Quick Start Kit
26. "GATT Fundamentals", Bluetooth SIG presentation, Bluetooth Smart Quick Start Kit

Δικτυακοί Τόποι

1. Bluetooth Technology Website: http://www.bluetooth.com
2. Bluetooth Technology Special Interest Group: https://www.bluetooth.org
3. Bluetooth Development Portal: https://developer.bluetooth.org
4. Bluetooth History: http://www.bluetooth.com/Pages/History-of-Bluetooth.aspx
5. Bluetooth GATT Definition Browser: https://developer.bluetooth.org/gatt/Pages/Definition-Browser.aspx
6. Bluetooth SIG Assigned Numbers: https://www.bluetooth.org/en-us/specification/assigned-numbers
7. Bluetooth Specification Adopted Documents: https://www.bluetooth.org/en-us/specification/adopted-specifications
8. Bluetooth XML Schemas: http://schemas.bluetooth.org/Pages/default.aspx

9. Bluetooth Smart Technology Overview: https://developer.bluetooth.org/TechnologyOverview/Pages/BLE.aspx
10. Bluetooth Custom Profile Development: https://developer.bluetooth.org/DevelopmentResources/Pages/Custom-Profile-Development.aspx
11. Bluetooth SIG Videos: https://www.youtube.com/user/BluetoothTech
12. Wikipedia – Bluetooth: http://en.wikipedia.org/wiki/Bluetooth
13. Wikipedia – Bluetooth Low Energy: http://en.wikipedia.org/wiki/Bluetooth_low_energy
14. Wikipedia – iBeacon: http://en.wikipedia.org/wiki/IBeacon
15. Fitbit Official Site: http://www.fitbit.com/home
16. Nike+ FuelBand SE: http://www.nike.com/us/en_us/c/nikeplus-fuelband

DA14580

Documentation

1. "DA14580 Datasheet v1.63", Dialog Semiconductor
2. "User manual – DA14580 Development Kit v1.1", Dialog Semiconductor
3. "User manual – DA14580 Software architecture v3.0", Dialog Semiconductor
4. "User manual – DA14580 Software Development Guide v1.1", Dialog Semiconductor
5. "User manual – DA14580 Sleep mode configuration v1.0", Dialog Semiconductor
6. "User manual – DA14580 Memory map and scatter file v1.0", Dialog Semiconductor
7. "User manual – DA14580 Peripheral Drivers v1.0", Dialog Semiconductor
8. "User manual – DA14580 Peripheral Examples v1.0", Dialog Semiconductor
9. "User manual – DA14580 Proximity application example in an integrated- and external-processor solution v1.0", Dialog Semiconductor
10. "DA14580 Development Kit Mainboard Schematic – Version C2.0", Dialog Semiconductor
11. "Application Note – DA14580 Supply Current Measurements v1.1", Dialog Semiconductor
12. "Application Note – DA14580-01 Cold boot timing and power details v1.0", Dialog Semiconductor

Specifications

1. "RivieraWaves Kernel (RW-BT-KERNEL-SW-FS) v1.1", RivieraWaves
2. "GAP Interface Specification (RW-BLE-GAP-IS) v2.04", RivieraWaves
3. "GATT Interface Specification (RW-BLE-GATT-IS) v2.00", RivieraWaves
4. "ATTDB Interface Specification (RW-BLE-ATTDB-IS) v1.0", RivieraWaves
5. "RW BLE Application Interface Specification (RW-BLE-APP-IS) v0.1", RivieraWaves
6. "RW-BLE Controller Software (RW-BLE-HOST-SW-FS) v1.0", RivieraWaves
7. "RW BLE Host Interface Specification (RW-BLE-HOST-IS) v1.0", RivieraWaves
8. "RW-BLE Host Software (RW-BLE-CTRL-SW-FS) v1.02", RivieraWaves
9. "RW BLE Battery Service Interface Specification (RW-BLE-PRF-BAS-IS) v2.0", RivieraWaves
10. "RW BLE Device Information Service Interface Specification (RW-BLE-PRF-DIS-IS) v2.0", RivieraWaves
11. "RW BLE Proximity Profile Interface Specification (RW-BLE-PRF-PXP-IS) v3.0", RivieraWaves
12. "RW BLE Find Me Profile Interface Specification (RW-BLE-PRF-FMP-IS) v3.0", RivieraWaves
13. "RW BLE Heart Rate Profile (HRP) Interface Specification (RW-BLE-PRF-HRP-IS) v0.4", RivieraWaves
14. "RW BLE Health Thermometer Profile Interface Specification (RW-BLE-PRF-HTP-IS) v3.0", RivieraWaves

15. “RW BLE Glucose Profile (GLP) Interface Specification (RW-BLE-PRF-GLP-IS_0) v0.3”, RivieraWaves
16. “RW BLE Blood Pressure Profile (BLP) Interface Specification (RW-BLE-PRF-BLP-IS) v0.5”, RivieraWaves
17. “RW BLE Time Profile (TIP) Interface Specification (RW-BLE-PRF-TIP-IS) v2.0”, RivieraWaves
18. “RW BLE Security Manager Interface Specification (RW-BLE-SM-IS) v1.1”, RivieraWaves

Δικτυακοί Τόποι

1. Dialog Semiconductor: http://www.dialog-semiconductor.com/
2. SmartBond DA14580 Bluetooth Smart: http://www.dialog-semiconductor.com/products/bluetooth-smart/smartbond-da14580
3. Keil Embedded Development Tools: http://www.keil.com/
4. Keil MDK-ARM: http://www.keil.com/arm/mdk.asp

Android

Βιβλία

1. Lauren Darcey, Shane Conder, “Android™ Wireless Application Development – Volume II: Advanced Topics, Third Edition”, Addison-Wesley, 2012
2. Joseph Annuzzi, Jr., Lauren Darcey, Shane Conder, “Introduction to Android™ Application Development – Android Essentials, Fourth Edition”, Addison-Wesley, 2013
3. Reto Meier, “Professional Android™ 4 Application Development”, John Wiley & Sons, 2012
4. Wei-Meng Lee, “Beginning Android™ 4 Application Development”, John Wiley & Sons, 2012

Δικτυακοί Τόποι

1. Android Developers: http://developer.android.com/index.html
2. Android Development Tools: http://developer.android.com/tools/index.html
3. Android API Guide – Bluetooth: http://developer.android.com/guide/topics/connectivity/bluetooth.html
4. Android API Guide – Bluetooth Low Energy:
5. http://developer.android.com/guide/topics/connectivity/bluetooth-le.html
6. Android Documentation – android.bluetooth package: https://developer.android.com/reference/android/bluetooth/package-summary.html
7. Android API Guide – Activities: https://developer.android.com/guide/components/activities.html
8. Android API Guide – Services: https://developer.android.com/guide/components/services.html
9. Android API Guide – App Widgets: https://developer.android.com/guide/topics/appwidgets/index.html
10. Wikipedia – Android: http://en.wikipedia.org/wiki/Android_%28operating_system%29
11. Wikipedia – Android version history: http://en.wikipedia.org/wiki/Android_version_history

Άλλη βιβλιογραφία

1. Brian W. Kernighan, Dennis M. Ritchie, "The C Programming Language, Senond Edition", Prentice Hall
2. Avi Silberschatz, Henry F. Korth, S. Sudarshan, "Database System Concepts, Fourth Edition", McGraw-Hill, 2002
3. "Cortex™-M0 – Technical Reference Manual", ARM, 2009
4. "Cortex™-M0 Devices – Generic User Guide", ARM, 2009
5. "I^2C-bus specification and user manual", NXP Semiconductors, 2014
6. "T5400 – Datasheet", EPCOS
7. "T5400 – Application note", EPCOS
8. "BMP180 – Datasheet", BOSCH
9. "HIH-6130/6131 – Datasheet", Honeywell
10. "HIH6130 – I^2C Communications", Honeywell
11. "HTU21D – Datasheet", MEAS
12. "Si1145/46/47 – Datasheet", Silicon Labs
13. "TSL4531 DIGITAL AMBIENT LIGHT SENSOR– Datasheet v2", TAOS
14. "TSL2560, TSL2561 LIGHT-TO-DIGITAL CONVERTER – Datasheet", TAOS
15. "Duracell cr2032 Datasheet", Duracell
16. "Energizer cr2032 Datasheet", Energizer
17. "Maxell cr2032 Datasheet", Maxell
18. "Panasonic cr2032 Datasheet", Panasonic

Δικτυακοί Τόποι

1. Wikipedia – FSK: http://en.wikipedia.org/wiki/Frequency-shift_keying
2. IETF – UUID: http://tools.ietf.org/html/rfc4122
3. SQLite: http://www.sqlite.org/
4. SQLite Datatypes: http://www.sqlite.org/datatype3.html
5. Wikipedia – UV Index: http://en.wikipedia.org/wiki/Ultraviolet_index
6. EPA – UV Index Scale: http://www2.epa.gov/sunwise/uv-index-scale