ICOM IC-9100 Bluetooth Control (IcomBTC)

Nombre Estudiante: Marcos Tomás Leirós Otero (EA5GQP)

Máster Universitario en Desarrollo de Aplicaciones para Dispositivos Móviles

Nombre Consultor/a: David Escuer Latorre
Profesor/a responsable de la asignatura: Carles Garrigues Olivella

Fecha de Entrega 05/06/2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Icom IC-9100 Bluetooth Control*
Nombre del autor:	*Marcos Tomás Leirós Otero*
Nombre del consultor/a:	*David Escuer Latorre*
Nombre del PRA:	*Carles Garrigues Olivella*
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	Máster Universitario en Desarrollo de Aplicaciones para Dispositivos Móviles
Idioma del trabajo:	*Castellano*
Palabras clave	Radio Bluetooth Control

Resumen del Trabajo (máximo 250 palabras):

Icom IC-9100 Bluetooth Control tiene como finalidad poder controlar vía Bluetooth desde el Smartphone un transceptor de radioaficionado de la Marca ICOM, concretamente el modelo de alta gama IC-9100. Mediante su empleo podemos visualizar y cambiar a nuestra voluntad los parámetros más importantes del equipo, como son la frecuencia, el modo de modulación, la potencia, la ganancia de micrófono, transmitir, etc.

El contexto de aplicación es el apasionando mundo de la radio, en su faceta de la radioafición. Los potenciales usuarios son por lo tanto radioaficionados que posean el equipo de referencia. Una anotación importante es que no solamente es interesante usarlo para operar nuestra radio desde la comodidad del sofá de nuestro salón o metidos en la bañera, sino que su uso puede ser de especial interés a personas con movilidad reducida.

Los resultados han sido incluso mejor que los esperados, ya que se ha convertido en una verdadera App que puede dar mucho de sí ya que muchos usuarios pueden estar interesados es trabajar con su radio desde otra habitación próxima al conocido como “cuarto de las chispas”, solucionándole por tanto IcomBTC la situación.

Las conclusiones son que solamente mediante la elaboración completa de un proyecto por parte del estudiante, desde la idea inicial hasta la obtención del resultado final, pasando por la planificación, el desarrollo, la depuración y las pruebas es posible integrar todos los conocimientos que componen el Master Oficial en Desarrollo de Aplicaciones para dispositivos Móviles de la UOC.

Abstract (in English, 250 words or less):

Icom IC-9100 Bluetooth Control has a powerful audience via Bluetooth from the smartphone a ham radio transmitter ICOM Brand, specifically the high-end model IC-9100. Using your mobile phone you can change and change the most important parameters of the equipment, such as frequency, modulation mode, power, microphone gain, transmit, etc.

The context of application is the passionate world of radio, in its facet of ham radio. Potential users are therefore radio amateurs who own the reference equipment. An important note is that it is not only interesting to use it to operate our radio from the comfort of the sofa in our living room or in the bathtub, but its use may be of special interest to people with reduced mobility.

The results have been even better than expected, since it has become a real App that can give much of itself as many users may be interested is working with your radio from another room next to the known as "room of sparks", thus solving IcomBTC the situation.

The conclusions are that only through the complete elaboration of a project on the part of the student, from the initial idea to the obtaining of the final result, going through the planning, the development, the purification and the tests it is possible to integrate all the knowledge that make up the Official Master in Application Development for Mobile Devices of the UOC.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

En los últimos tiempos ha cambiado mucho el contexto en el mundo de la radio-afición con la aplicación de las nuevas tecnologías, ahora los equipos de radio amateur suelen llevar algún conector de comunicación serial que por medio del envío y recepción de comandos específicos (cadenas de caracteres) permiten su control, bien por medio del ordenador PC en modo local o bien por internet. En éste último sentido hoy día vienen numerosos equipos que además del puerto serie llevan incorporado conexión Ethernet, aunque debido a su elevado precio para el control remoto lo que más abunda es usar una computadora conectada al puerto serie y que actúe como servidor.

Ilustración 1: President Lincoln I y II

Este Trabajo Final de Máster se justifica debido a la utilidad de poder emplear un equipo transceptor de radioaficionado desde una habitación distinta pero próxima a la de su ubicación física. En efecto, mediante nuestro Smartphone podemos operar nuestro equipo de radio desde la comodidad del sofá del salón o incluso desde la bañera. Sin olvidarnos de que es de extremo interés para su uso por parte de personas con movilidad reducida.

En la actualidad no existen App's con tal finalidad, ya que las emisoras de radioaficionado no llevan incorporado dispositivo Bluetooth para poder controlarla, no obstante si existen aplicaciones de escritorio y móviles para controlar el equipo desde Internet, la cuales podremos emplear como inspiración. Es por esta causa que se justifica la necesidad en el contexto de la radioafición de una App que haga las veces de mando a distancia vía BT y que consiga las delicias de los radioaficionados.

1.2 Objetivos del Trabajo

El objetivo de éste TFM o proyecto final es **desarrollar en Java una App para Android mediante Android Studio que sea capaz de gestionar vía Bluetooth los parámetros más usuales de un equipo de radioaficionado** especialmente la frecuencia y el modo de modulación. Concretamente de un equipo transceptor marca ICOM modelo IC-9100, es un equipo que tiene todas las bandas de radio HF/VHF/UHF (De 30Khz a 1320Mhz en todos los modos).

Ilustración 2: Icom IC-9100

Como consecuencia del anterior, aparece un nuevo objetivo que es **diseñar e implementar un interface SERIAL <-> BT** que permita enlazar de manera bidireccional la comunicación vía Bluetooth del dispositivo móvil con la comunicación vía Serial del equipo de radio.

El motivo de la necesidad de éste hardware es debido a que el emisor/receptor Icom IC-9100 no lleva Bluetooth incorporado, pero si un puerto serie destinado a su posible control mediante PC, por tanto hay que adaptar las señales entre el conector serie y el Bluetooth del teléfono móvil o Tablet.

1.3 Enfoque y método seguido

Como estrategia de diseño hemos escogido elaborar una App totalmente nueva desde cero, ya que no existe que se sepa otra App con la misma funcionalidad pretendida y menos con el código fuente disponible para su adaptación.

No obstante durante el proceso de búsqueda de información se han encontrado tres artículos de desarrollo Android con código fuente (un ejemplo de uso del BT,

un ejemplo de dial de tipo perilla y un custom bar) disponible los cuales una vez evaluados y probados veo que son ideales como punto de partida. Tenemos toda la información sobre éstos ejemplos más adelante por ello no comentamos más en éste punto.

Por tanto la estrategia a emplear para implementar la App nueva es aprovechar parte del código de éstos tres ejemplos y adaptarlo para que nos valga como código base del TFM.

Por otro lado nos decantamos por una App para la plataforma Android, debido a que la aplicación está destinada a un tipo de usuarios muy concretos, radioaficionados de todo el mundo, y Android es la plataforma más empleada, especialmente para software libre.

1.4 Planificación del Trabajo

A continuación de describen los ítems inicialmente reconocidos al efectuar el estudio preliminar de la planificación del trabajo final de máster, seguido de su correspondiente Diagrama de Gantt, elaborado con el prestigioso software Microsoft Project. No obstante el diagrama podría ser modificado durante el desarrollo del proyecto debido a diversas causas, problemas técnicos, aparición de nuevos ítems no reconocidos previamente, indisposición del desarrollador etc.
Debido a mi vida familiar y trabajo preveo trabajar en días laborables a razón de 4h/día.
No obstante lo anterior en caso de bloqueo o retraso del proyecto debido a lo anteriormente enumerado u otras causas se prevé aumentar el tiempo usando algunos fines de semana para solventar los problemas técnicos o nuevos ítems, siempre con el objetivo de terminar dentro del plazo dado, ya que todo proyecto tiene que contar con planes de contingencia con el fin de ajustarse a los plazos.

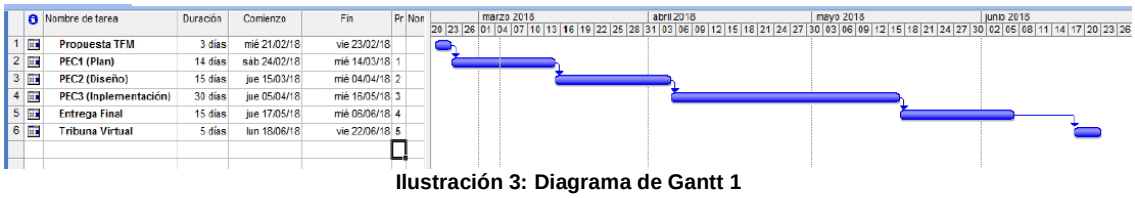
Ítems generales
- Estudio inicial y propuesta de TFM
- PEC1 (Plan de trabajo)
- PEC2 (Diseño)
- PEC3 (Implementación)
- Entrega Final
- Tribunal Virtual

Ítems complementarios (dentro de los generales)
- Contextualización y diseño (DCU)
- Wireframes y mockups
- Configuración del módulo HC-06 mediante comandos AT Hayes
- Diseño del interface Serial <-> Bluetooth (esquema electrónico)
- Pruebas del interface en protoboard
- Fabricación del PCB del interface
- Montaje del interface
- Diseño de scenes de la App
- Diseño del dial del tipo perilla, retoque de fotos
- Programación inicial, pruebas BT, conversión ascii/hex…

- Programación refinada
- Pruebas y testeo
- Elaboración de la memoria final del TFM

Diagrama de Gantt

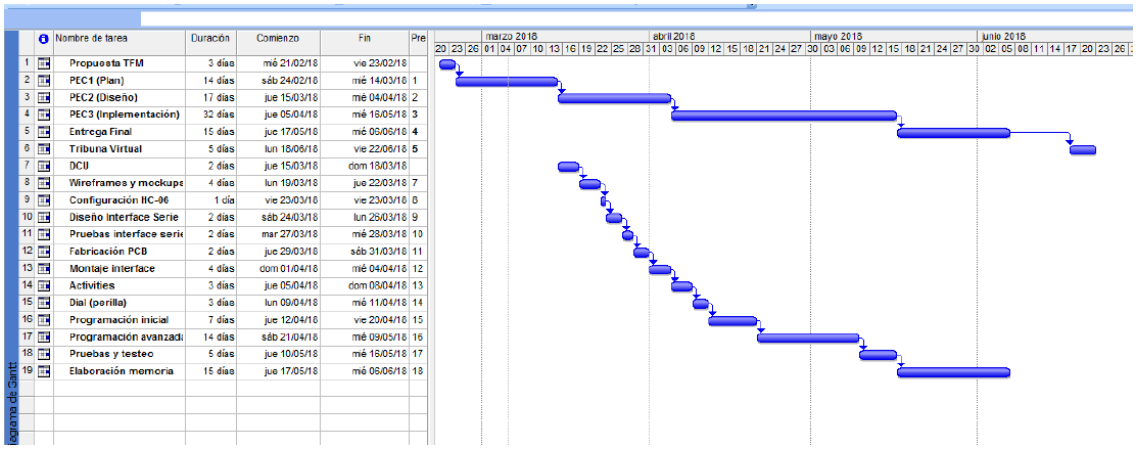
Primero vemos los ítems generales para cumplir los plazos de la asignatura.



	Nombre de tarea	Duración	Comienzo	Fin	Pr	Nom
1	Propuesta TFM	3 días	mié 21/02/18	vie 23/02/18		
2	PEC1 (Plan)	14 días	sáb 24/02/18	mié 14/03/18	1	
3	PEC2 (Diseño)	15 días	jue 15/03/18	mié 04/04/18	2	
4	PEC3 (Inplementación)	30 días	jue 05/04/18	mié 16/05/18	3	
5	Entrega Final	15 días	jue 17/05/18	mié 06/06/18	4	
6	Tribuna Virtual	5 días	lun 18/06/18	vie 22/06/18	5	

Ilustración 3: Diagrama de Gantt 1

Y seguidamente añadimos los complementarios pero necesarios para poder completar los generales



	Nombre de tarea	Duración	Comienzo	Fin	Pre
1	Propuesta TFM	3 días	mié 21/02/18	vie 23/02/18	
2	PEC1 (Plan)	14 días	sáb 24/02/18	mié 14/03/18	1
3	PEC2 (Diseño)	17 días	jue 15/03/18	mié 04/04/18	2
4	PEC3 (Inplementación)	32 días	jue 05/04/18	mié 16/05/18	3
5	Entrega Final	15 días	jue 17/05/18	mié 06/06/18	4
6	Tribuna Virtual	5 días	lun 18/06/18	vie 22/06/18	5
7	DCU	2 días	jue 15/03/18	dom 18/03/18	
8	Wireframes y mockups	4 días	lun 19/03/18	jue 22/03/18	7
9	Configuración HC-06	1 día	vie 23/03/18	vie 23/03/18	8
10	Diseño interface Serie	2 días	sáb 24/03/18	lun 26/03/18	9
11	Pruebas interface serie	2 días	mar 27/03/18	mié 28/03/18	10
12	Fabricación PCB	2 días	jue 29/03/18	sáb 31/03/18	11
13	Montaje interface	4 días	dom 01/04/18	mié 04/04/18	12
14	Activities	3 días	jue 05/04/18	dom 08/04/18	13
15	Dial (perilla)	3 días	lun 09/04/18	mié 11/04/18	14
16	Programación inicial	7 días	jue 12/04/18	vie 20/04/18	15
17	Programación avanzadi	14 días	sáb 21/04/18	mié 09/05/18	16
18	Pruebas y testeo	5 días	jue 10/05/18	mié 16/05/18	17
19	Elaboración memoria	15 días	jue 17/05/18	mié 06/06/18	18

Ilustración 4: Diagrama de Gantt 2

1.5 Breve sumario de productos obtenidos

Básicamente los productos a obtener son dos, la implementación final de la App Android propiamente dicha y el interface hardware Serial<->BT, siendo este segundo producto algo alejado de los objetivos del máster.

EL INTERFACE:
Se va a basar como pieza angular en el módulo HC-06 muy usado en el mundo de Arduino. No obstante es obvio que tendrá componentes añadidos ya que por un lado se alimenta a 5Vdc cuando de la emisora podemos obtener 12Vdc por lo que hay que reducir la tensión, por otro lado la comunicación de la emisora va a dos hilos (Tx-Rx/-) pero el módulo BT a tres (Tx/Rx/-) y hay que convertirlo. El diseño, implementación y pruebas vendrá en las siguientes PEC.

App "ICOM IC-9100 BLUETOOTH CONTROL":
Es la App en sí misma. Se deberán solucionar numerosos problemas, como la comunicación BT bidireccional, la conversión de datos o comandos ASCII<->Hex, el diseño del dial (la perilla de sintonía) detectando gestos, detección y corrección de errores, etc.

2. Diseño del proyecto e interface.

2.1. Diseño centrado en el usuario DCU.

2.1.1) Usuarios y contexto de uso (Persona focal/secundaria/excluida, Escenarios)

2.1.1.a) Ficha de la persona focal

Estamos hablando de radioaficionados de todo el mundo que usan transceptores de radio de la marca ICOM (http://www.icomspain.com/), especialmente si es el modelo IC-9100.

Pepe Pardo está casado y tiene dos hijos mayores, ya que tiene 67 años. Vive en Oropesa del Mar un pequeño pueblo marinero al norte de la capital de Castellón, su residencia está junto al mar por lo que toda la vida ha estado en un ambiente sanamente respirable. Ha trabajado de profesor de tecnología, ahora es jubilado aunque continúa ayudando 2h al día por la tardes en la escuela de adultos de la localidad impartiendo física y matemáticas, así se obliga a estar en constante movimiento y esto le ayuda a estar en forma tanto física como mental, ya que también le vale parar relacionarse con los demás.

Debido a no pertenecer a la “generación digital” tiene una brecha en este sentido comparado con sus nietos, aunque ellos le ayudan en lo que pueden. Es muy tranquilo y le gusta leer en un sofá unipersonal que se ha comprado con un diseño especial para la lectura. Tiene un grupo de amigos con los que queda al menos un fin de semana al mes para salir de excursión por las cercanías, las últimas rutas han sido subir al Monte Bartolo (Benicasim) desde la Ermita de Les Santes (Cabanes) en el Paraje Natural del Desierto de las Palmas y una subida al Peñagolosa en el interior de Castellón ya cerca de la provincia de Teruel y que es considerado el pico más alto de la Comunidad Valenciana con sus orgullosos 1813 metros de altitud.

Cuando no está en la escuela de adultos o con los amigos ocupa todo su tiempo con la radio (su mujer tristemente ya falleció), ya que es radioaficionado de toda la vida, tiene licencia de radioafionado desde los 15 años dada por el Ministerio de Industria tras pasar unos exámenes de legislación, radioelectricidad y manejo de equipos de radio. Es un auténtico forofo de la radio, y está ilusionado con los equipos de la marca ICOM, concretamente uno de ellos es un IC-9100 (como se ve en la foto). Le iría bien un mando a distancia para la emisora ya que así al tener tantos equipos podría operar varias a la vez una con la mano ya que la tiene a distancia y la otra con el teléfono móvil. Sabe de tecnología y no tendrá problemas.

2.1.1.b) Ficha de la persona secundaria

Se trata de una persona similar a la primera pero con movilidad reducida, que bien podría ser considerada focal al tener las mismas inquietudes, pero la considero secundaria al ser muchos menos y por tanto será un pequeño porcentaje del bosque de usuarios.
Stephen Falken está casado y tiene tres hijos mayores, ya que tiene 76 años. Vive en América, y es un científico reconocido, por tanto tiene grandes conocimientos tecnológicos. Hasta media vida no conocía el mundo de la radioafición, aunque siempre se había visto atraída por él.

Una vez supero cierta edad comenzó a tener tiempo libre en casa, por lo que un amigo le introdujo en este apasionante mundo hasta que se ha convertido en un gran usuario y un excelente operador de radio.
Y como su amigo usa equipos de la marca más reconocida internacionalmente ICOM INC (Japón) él se ha sentido atraído por la misma, y posee numerosos equipos de dicha marca, como el IC-9100 (https://www.universal-radio.com/catalog/hamhf/5091.html) o el fabuloso receptor IC-R9500 entre otros.

Debido a su movilidad reducida le iría de maravilla poder operar el equipo de radio sin necesidad de tocarlo físicamente, y como muestra app ICOM Bluetooth Control (ICOM_BTC) va a funcionar tanto con teléfono móvil como tablet le iría como anillo al dedo para poder usarla desde la silla de ruedas.

2.1.1.c) Ficha de la persona excluida

Toda aquella persona que no es radioaficionada y por tanto no tiene ningún transceptor de radio, ya que esta app no la podría ni probar. Se incluyen en esta categoría los radioaficionados pero que no tienen equipos de la marca ICOM, normalmente porque son forofos de otras marcas de radio, como YAESU (https://www.yaesu.com/) o KENWOOD (http://www.kenwood.es/comm/).

Tomás Escribano es un joven de Tortosa y tiene 24 años, estudia en la UOC un Máster, ya que acaba de terminar un Grado en Deportes y quería especializarse por medio de una buena formación de postgrado, es muy deportista. Los fines de semana trabaja de camarero extra en el Parador Nacional enclavado en el Castillo de la Suda para pagarse sus estudios ya que la universidad no es barata, por lo que siempre está en movimiento. Debido a la edad y a su ritmo de vida NO tiene ningún interés en la radio ni mucho menos en la radioafición, nunca ha tenido un transceptor de radio, incluso opina que están “un poco locos”.

2.2) Diseño conceptual

2.2.a) Escenario 1

Pepe tiene el equipo de radioaficionado instalado en una pequeña oficina en su piso (lo que conocemos como el cuarto de las chispas) y sus hijos que le visitan a veces se quejan de que no sale de allí ni cuando está con ellos, por eso ha conectado un pequeño transmisor de radio fm (de esos que se usan mucho ahora en los automóviles para oír los mp3 por el propio coche) al audio de la emisora Icom, así puede escuchar a sus amigos radioaficionados por medio de la radio receptor fm del salón comedor en una frecuencia libre. El problema es que no puede cambiar de frecuencia o canal, por tanto le va de maravilla un mando a distancia desde su propio Smartphone vía Bluetooth para cambiarla sin desplazarse, la App Icom IC-9100 BT Control le ha solucionado la papeleta y mejorado su vida social.

Ilustración 5: Mini Emisor de Radio FM

2.2.b) Escenario 2

Stephen quiere operar su maravillosa y flamante Icom IC-9100, pero hoy NO tiene asistente en casa, es su día libre. ICOM Bluetooth Control está aquí para solucionarle el problema y facilitarle la vida, ya que sentado en su silla de ruedas puede operar tocando en la pantalla de la Tablet que tiene en su mini-escritorio encima de la rodilla (al otro lado del ordenador parlanchín).

Ilustración 6: Ordenador parlanchín

2.3) Prototipado de alta fidelidad (MockUp)

Empleamos para tal fin el software Justinmind Prototyper. Se muestra a continuación el **WIREFRAMING** (Árbol de navegación). La idea inicial es una App de tipo HUB, desde donde cualquier layout se pueda regresar al layout principal.

Diagrama de navegación inicialmente propuesto.

Ilustración 7: Diagrama de navegación

2.4) Diseño de la arquitectura y singularidades

Usaremos el paradigma MVC, Modelo Vista Controlador. De manera que en la pantalla principal (main) se mostrarán los tres datos más relevantes en un equipo de radio, la frecuencia en la que estamos en Hz, el modo de modulación (lsb/usb/cw/rtty/am/fm/dv) y la señal del medidor smeter.

Para conseguir esto tendremos un controlador que solicitará vía comando serie estos tres parámetros de forma repetitiva cada tres segundos, seguidamente analizará la cadena de caracteres que componen la respuesta al comando, y actuará en consecuencia para mostrarlos en la pantalla.

Por otro lado tanto la frecuencia como el modo de modulación las podremos modificar a nuestro gusto, así como otras propiedades.
Un aspecto muy importante de diseño es la perilla o dial de sintonía.

Ilustración 8: Perilla rotatoria o Knob real

Podemos observar siguiendo la flecha la perilla de sintonía (rotary knob control) del IC-9100 para subir y bajar la frecuencia, En la app de pondrá una que realmente será una foto, y se hará rodar con un gesture detector al detectar si deseamos un lado u otro, de manera que calcule la frecuencia a mandar y se envíe por el bluetooth en forma de comando, analizado la respuesta para ver si ha sido exitosa o no.
Para conseguir esto sin “parar” la app tendremos que usar un thread, runnable, handler o asyntask.

Buscando en internet hemos encontrado un ejemplo de una perilla (knob) que nos puede valer como base para modificarla a nuestro gusto, seguidamente vemos su aspecto inicial. Es evidente que requerirá dar vueltas completas, detectar la dirección de giro (no los grados como hace ahora), y modificar el aspecto de los drawables.

Ilustración 9: Knob

Otra singularidad es el medidor de señal en dB's o s-meter. Se puede usar un gadget o bien cualquier elemento que podamos cambiar el color, siendo los primeros verdes los segundos amarillos y los últimos rojos para marcar la máxima señal. Vemos un mediador s-meter digital real.

Ilustración 10: S-meter

Buscando de nuevo en la red observamos un diseño que adaptado nos será útil, se trata de un custom seek bar, vemos el aspecto.

Ilustración 11: Custom Seek bar

Por otro lado tenemos elementos de control que originalmente en la radio son también rotatorios pero que nosotros decidimos usar seekbars, el motivo de tal decisión es que para el dial de sintonía si es casi obligado usar una perilla giratoria para darle "realidad" pero para otros mandos como el volumen, la potencia, el silenciador squelch y la ganancia de micrófono es más cómodo deslizar el dedo, además no queremos llenar la pantalla de un pequeño Smartphone de diales que ocupan demasiado espacio. Vemos los mandos originales.

Ilustración 12: Otros mandos rotatorios

Nosotros usaremos para estos mandos un seekbar standard.

Ilustración 13: SeekBar

Más adelante veremos las clases y los métodos necesarios.

2.5 Interface Hardware (Bluetooth Full Dúplex <-> Serial TTL Half Dúplex)

2.5.1) Diseño del esquema

Abordamos ahora el diseño y construcción del interface mencionado anteriormente, ya se dejó claro que la emisora no tiene interface BT por lo que vamos a añadir uno, escogemos el HC-06 del mundo Arduino por su bajo precio (5€) y su facilidad de encontrar.

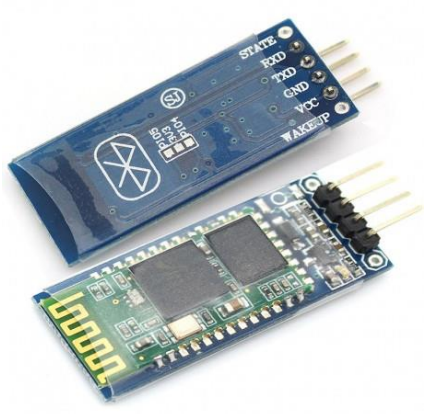


Ilustración 14: HC-06

Por otro lado tenemos las particularidades eléctricas de que dicho módulo tiene tres pines para la comunicación un TX, un RX (full dúplex) y masa o GND, sin embargo el transceptor de radio lleva incorporado un conector (16 en la siguiente foto) conocido como CI-V de solamente dos hilos, uno para TX-RX (half-dúplex) y la masa. Además el módulo BT requiere una alimentación de 5VDC mientras que la señal que sale de la emisora (12 en la siguiente foto) es de 12VDC.

Ilustración 15: IC-9100 Panel trasero

Requerimos pues diseñar un interface que “mezcle” de alguna manera tx/rx y reduzca la tensión al mismo tiempo.

El esquema queda así (La emisora entrega 12v por las patillas 2/8 del conector Acc-13, y el conector CI-V es de tipo jack mono de 3’5mm):

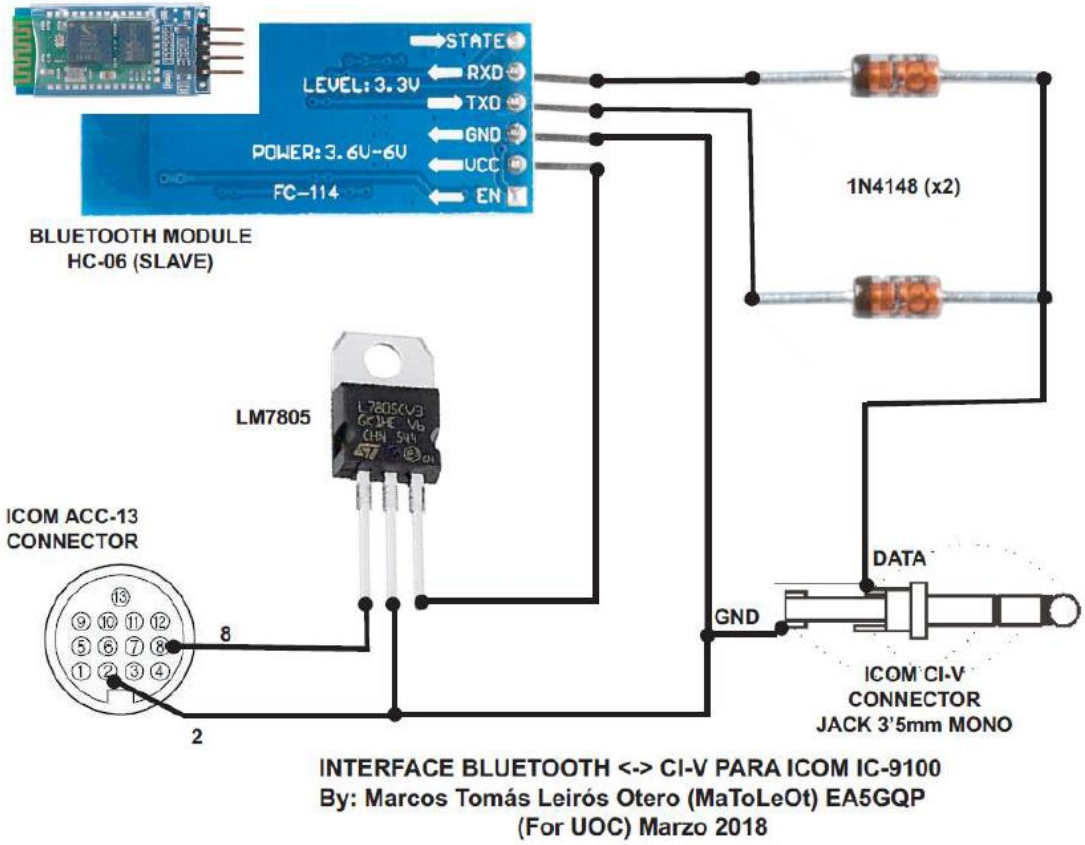


Ilustración 16: Esquema del Interface

MATERIAL

- Módulo Bluetooth esclavo HC-06
- Estabilizador 7805 (Encapsulado TO220)
- 2x Diodos 1N4118
- Jack mono de auriculares 3’5mm

- Conector ACC-13 (Suministrado por ICOM con la emisora)
- Cables
- Placa PCB de circuito impreso
- Caja de ABS

2.5.2) Diseño del circuito impreso (PCB)

Una vez validado el diseño electrónico pasamos al dibujo por computador de una placa de circuito impreso fija PCB. El software escogido es el obsoleto pero excelente TANGO PCB PLUS de Ms-Dos, que cómo no funciona desde que salió windows 7 se usa por medio de una máquina virtual (VirtualBox) con windows XP instalado dentro.

Antes que nada creamos la placa del tamaño apropiado para que entre en una caja de ABS de la marca supertronic de tamaño 67x67x23mm, y seguidamente situamos componentes y vamos dibujando pistas.

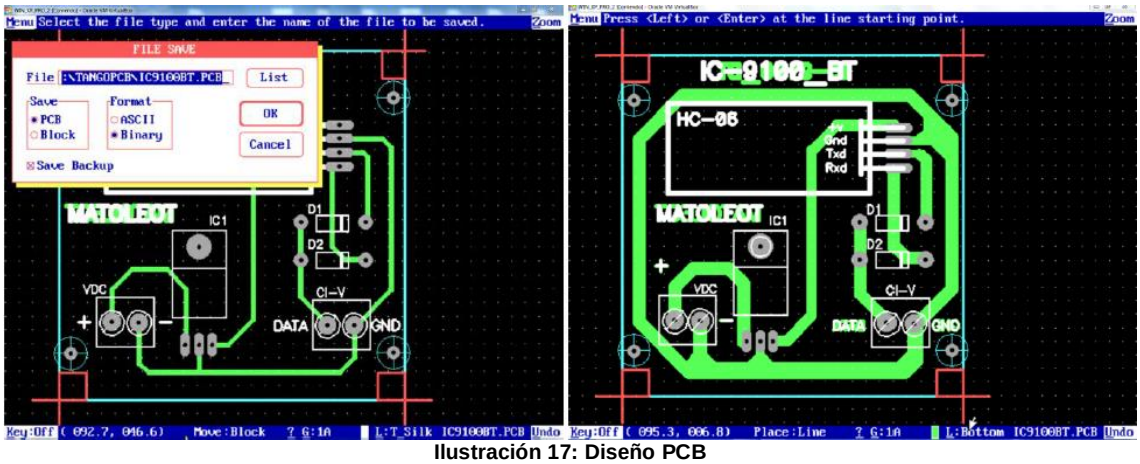


Ilustración 17: Diseño PCB

El resultado final es el siguiente:

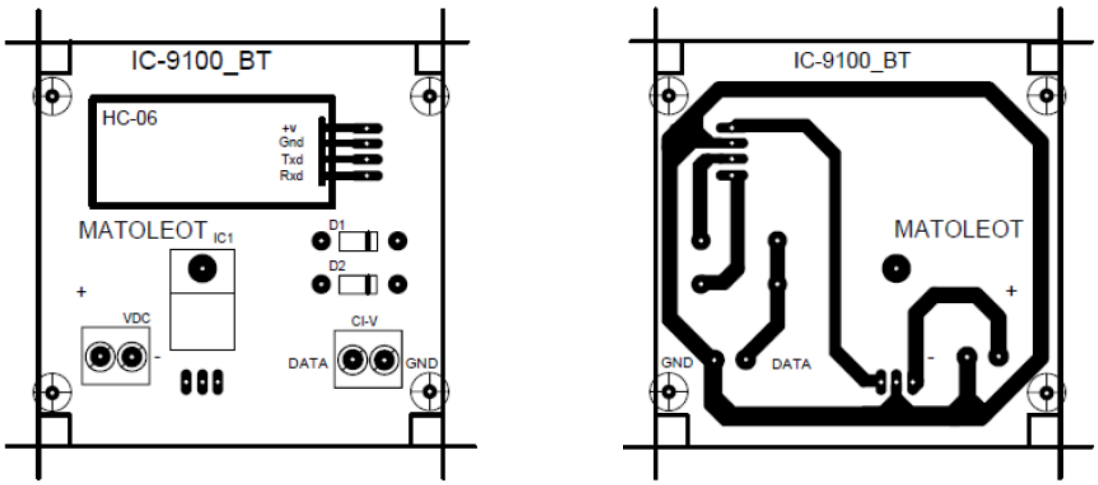


Ilustración 18: PCB final

2.5.3) Fabricación del circuito impreso (PCB)

Usamos el método de la insoladora para conseguir un buen acabado del circuito, por lo tanto imprimimos el diseño mediante una lámina de acetato.

Vemos la insoladora con el fotolito puesto.

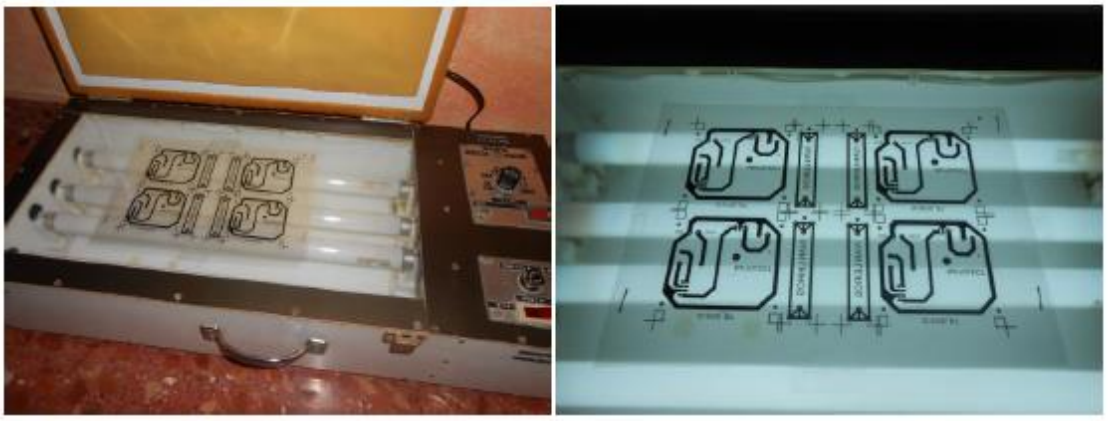


Ilustración 19: Insoladora y Fotolito

Ilustración 20: Revelador y acido

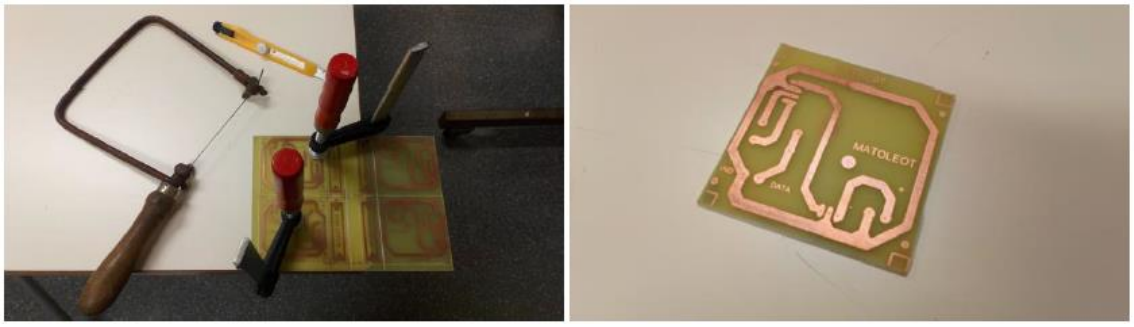


Ilustración 21: Recorte PCB

NOTA: El revelador es sosa cáustica rebajado con agua (mejor comprarlo en tienda de electrónica). El ácido es 25% de salfumant, 25% de agua oxigenada de 110 volúmenes, 50% de agua.

2.5.4) Montaje final

Taladrado

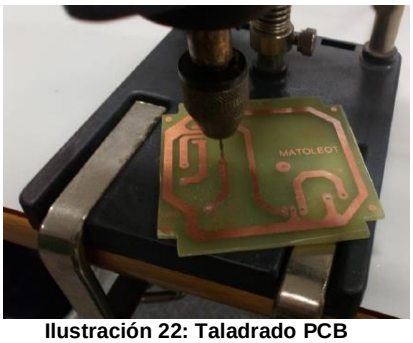


Ilustración 22: Taladrado PCB

Soldado

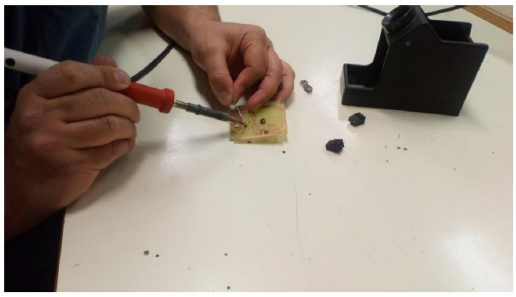


Ilustración 23: Soldadura PCB

Aspecto final del PCB soldado.

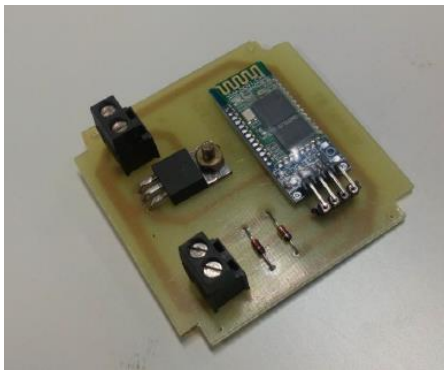


Ilustración 24: Interface terminado 1

Interface ya terminado y en su caja.

Ilustración 25: Interface terminado 2

2.5.5) Configuración del módulo Bluetooth. Comandos AT Hayes.

El módulo HC-06 viene de fábrica con parámetros por defecto en cuanto a su nombre, clave, velocidad etc. Para cambiarlos se hace mediante comandos Hayes AT, intercalando un Arduino como intermediario.

Vamos a dejarlo en **19200bps** (máximo de la emisora), de nombre de dispositivo le ponemos **ICOM_IC-9100_CI-V** y de password **9100**.

Conectamos el módulo al Arduino como en la figura (realmente usamos un Arduino Mega no el Uno como en la siguiente imagen).

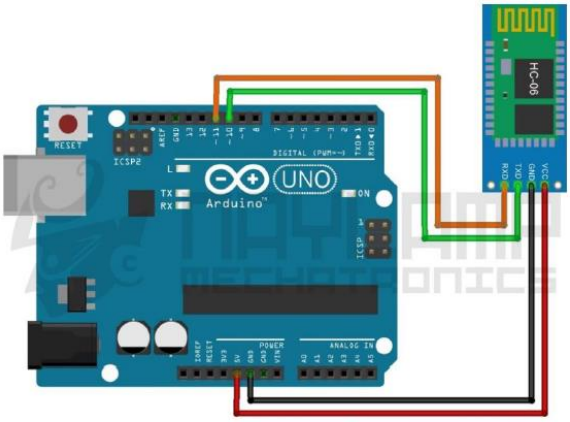


Ilustración 26: HC-06 con arduino

Ahora mediante el Arduino IDE insertamos un código que hará de puente entre los comandos AT que vamos a mandar dese el PC y el módulo BT.

Ilustración 27: Programa arduino

Sin conectar ningún dispositivo por BT le mando por la consola serie del programa Arduino IDE los comandos, analizando si la respuesta que nos da es correcta.

Ejemplos de Comandos AT para el BT HC-06

AT: check the connection
AT+NAME: Change name. No space between name and command.
AT+BAUD: Change baud rate, x is baud rate code, no space between command and code.
AT+PIN: change pin, xxxx is the pin, again, no space.
AT+VERSION

Le cambiamos el nombre a ICOM_IC-9100_CI-V

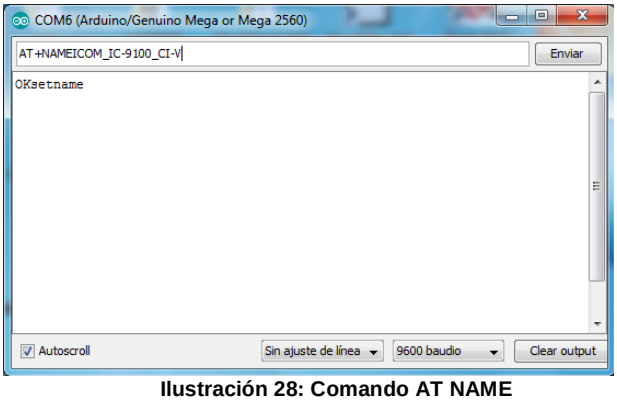


Ilustración 28: Comando AT NAME

Le cambiamos el password a 9100

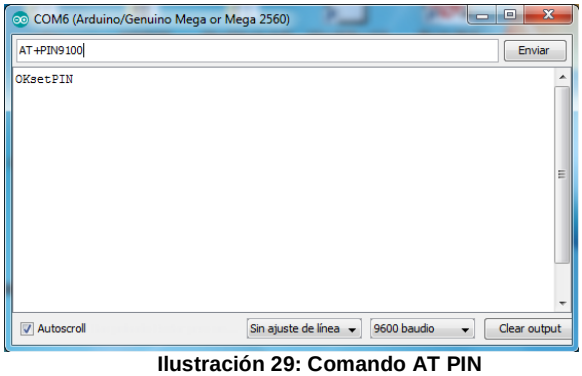


Ilustración 29: Comando AT PIN

Seguidamente mediante AT+BAUD5 lo cambiamos a 19200bps.

Con estos cambios queda el módulo Bluetooth HC-06 listo para su uso y con los parámetros deseados.

2.6) El protocolo serie CI-V.

Una vez tenemos el interface BT conectado a la emisora y configurado, para comenzar el desarrollo del programa en si necesitamos antes saber el funcionamiento del protocolo empleado por las emisoras ICOM, ya que en caso contrario no sabríamos que programar.

ICOM emplea en todos su equipos de radio el protocolo conocido como CI-V
http://www.icom.co.jp/world/support/download/manual/pdf/IC-7610_ENG_CI-V_0a.pdf

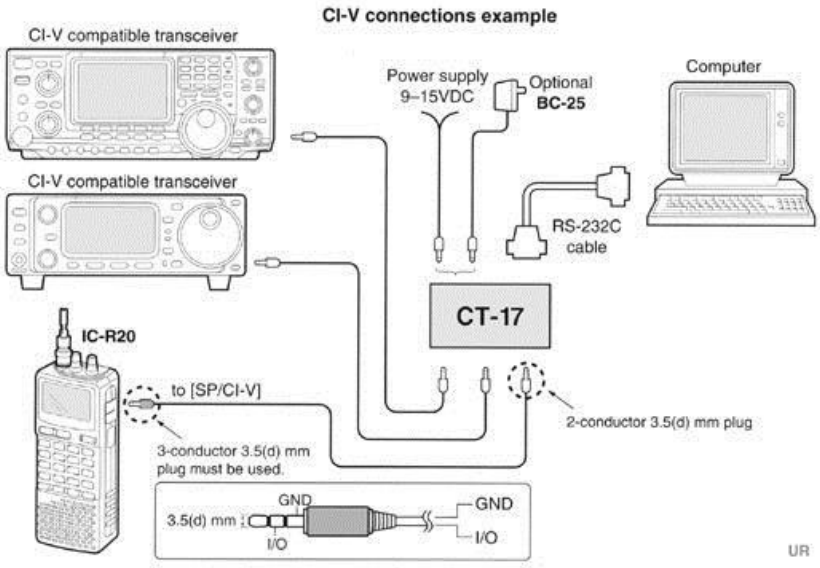


Ilustración 30: Icom CT-17

Eléctricamente cada carácter no es mas que una comunicación serie SSP de toda la vida con características 8/N/1 (8 bits de datos, sin paridad, 1 de stop). Admite hasta 19200bps.

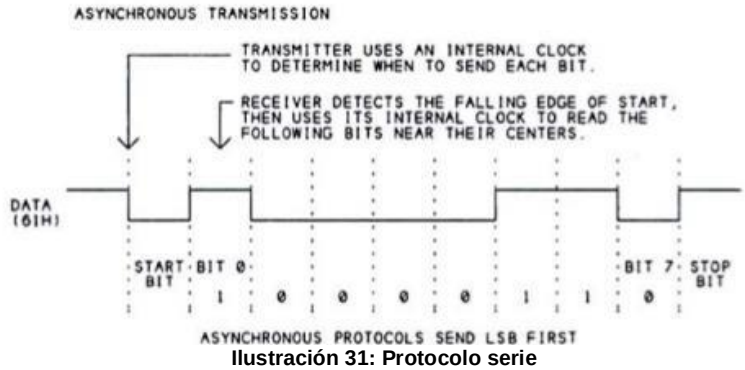
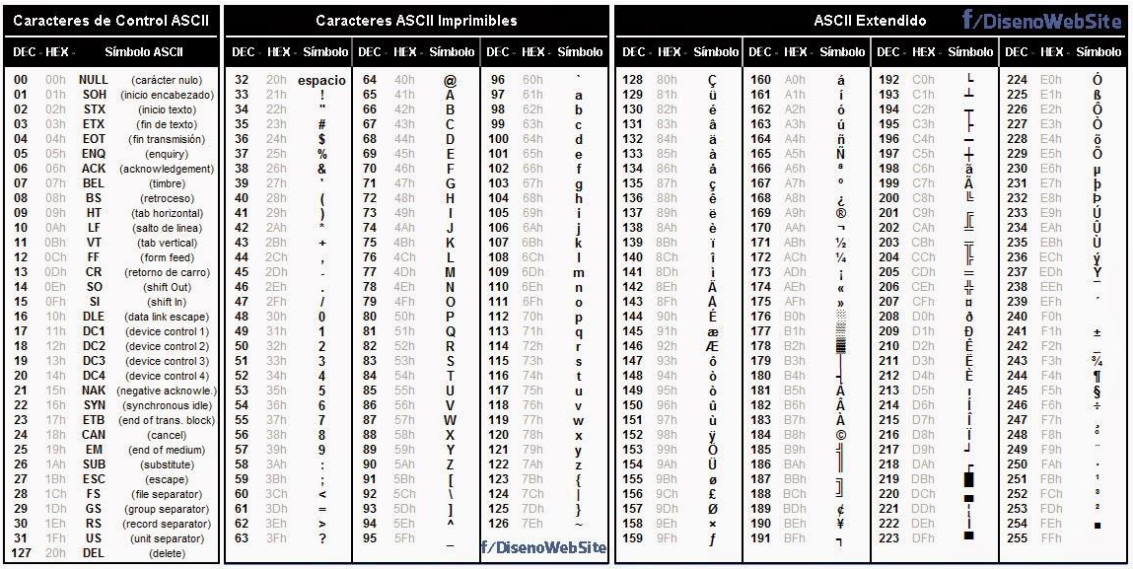


Ilustración 31: Protocolo serie

Cada carácter a enviar es uno de la tabla ASCII, pero en representación hexadecimal. Las cadenas en hex ocupan la mitad de bits que en string.



Caracteres de Control ASCII

DEC	HEX	Símbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

Caracteres ASCII Imprimibles

DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

ASCII Extendido

DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	á	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	▀	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	▌	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	▐	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

f/DisenoWebSite

Ilustración 32: Tabla ASCII 8 bits

El protocolo en si tiene como base estas características, interesa fijarse en la cabeza y fin.

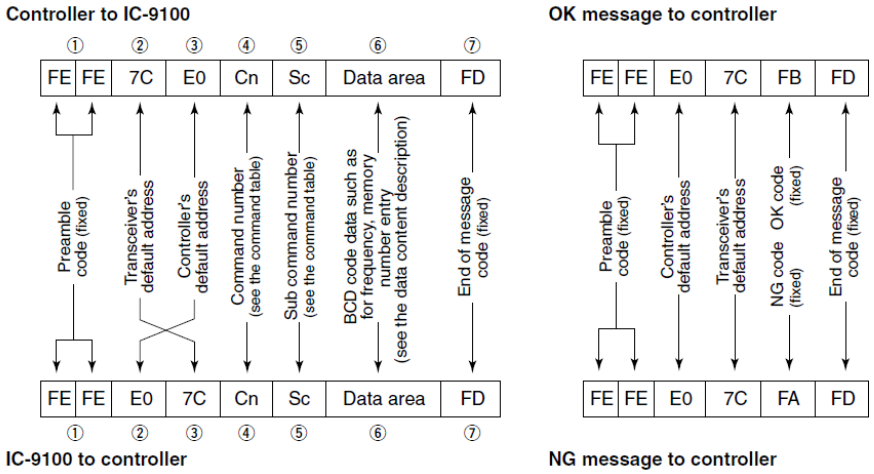


Ilustración 33: Protocolo CI-V 1

Como vemos cada comando tiene como cabeza FEFE y como fin FD (dos caracteres 254 y otro carácter 253 de la tabla ASCII). El E0 siempre se transmite y delimita el 7C que es la dirección por defecto (aunque configurable) de la emisora IC-9100 tomada como base en este proyecto, esa dirección tiene que coincidir en el programa de comunicación y en el equipo.

Por tanto para transmitir un comando siempre comenzará por FEFE7CE0 y siempre deberemos esperar que la vuelta comience por FEFEE07C. Como vemos el comando de regreso FB significa ACK y el FA NAK, pero el comando de vuelta puede ser mucho más complejo, veamos el de recibir la frecuencia, el comando 03.

◇ Command table

Cmd.	Sub cmd.	Data	Description
00		see p. 190	Send operating frequency for transceive
01		see p. 190	Send operating mode for transceive
02		see p. 191	Read band edge frequencies
03		see p. 190	Read operating frequency
04		see p. 190	Read operating mode
05		see p. 190	Send operating frequency
06		see p. 190	Send operating mode
07			Select VFO mode
	00		Select VFO A

Ilustración 34: Protocolo CI-V 2

Su respuesta sería (5 caracteres hex, 10 en ASCII)…

• Operating frequency

Command : 00, 03, 05

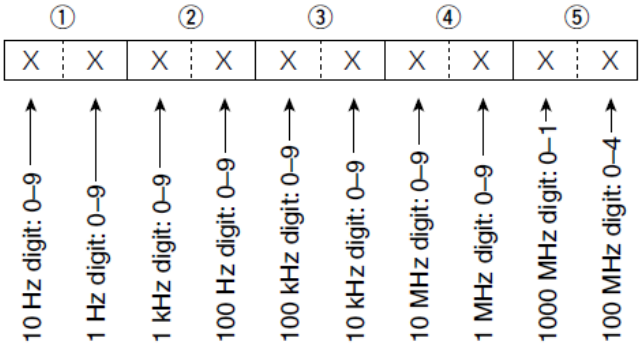
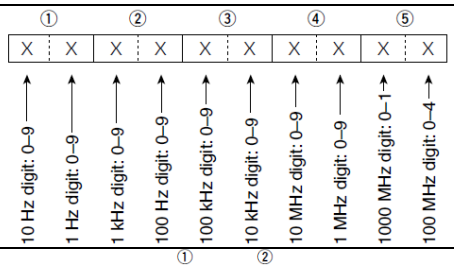
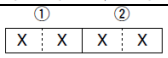


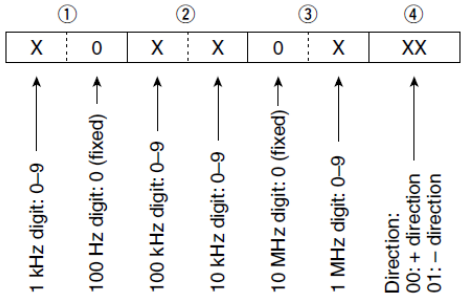
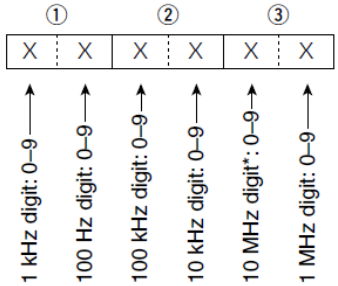
Ilustración 35: Protocolo CI-V 3

Vamos que aparte de usar siempre 10 caracteres (5 en hex) no vienen en orden, por lo que nuestro código tiene que formatearlo.

Elaboramos en este punto una tabla con los comandos que se han usado en la elaboración de este proyecto entre todos los disponibles, y la respuesta que se espera de ellos y que analiza nuestra app mara mostrar una información u otra en la pantalla, el comando de la tabla sustituye a la “??”.

TX (FEFE7CE0??FD) RX (FEFEE07C??FD)

LABOR	TX	RX	INTERPRETACIÓN DE DATOS
Frecuencia en auto (la manda la emisora si cambia manualmente)	-	00XXXXXXXXX	
Modo en auto (lo manda la emisora si cambia manualmente)	-	01XXXX	 ① Operating mode: 00: LSB, 01: USB, 02: AM, 03: CW, 04: RTTY, 05: FM, 07: CW-R, 08: RTTY-R, 17: DV; ② Filter setting: 01: FIL1, 02: FIL2, 03: FIL3
Lee la frecuencia	03	03XXXXXXXXX	Idem comando 00
Lee el MODE	04	04XXXX	Idem comando 01
Envía la frecuencia	05XXXXXXX	FB (ok) or FA (ng)	Idem comando 03

	XXX		
Envía el MODE	06XXXX	FB or FA	Idem comando 04
Pasa a VFOA	0700	FB or FA	-
Pasa a VFOB	0701	FB or FA	-
Modo CH	08	FB or FA	-
Envía CH	08xxxx	FB or FA	0001 to 0105: Select Memory channel (0001=M-CH01 to 0099=M-CH99, 0100=1A, 0101=1b, 0102=2A, 0103=2b, 0104=3A, 0105=3b)
Lee Split/Offset	0F	0FXX	Read Split function or duplex setting (00=OFF, 01=ON, 11=DUP–, 12=DUP+)
SPLIT a off	0F00	FB or FA	-
SPLIT a on	0F01	FB or FA	-
SIMPLEX	0F10	FB or FA	-
-DUP a ON	0F11	FB or FA	-
ANNOUNCE	1300	FB or FA	-
Lee VOLUMEN	1401	1401XXXX	0000 to 0255: Send/read [AF] position (0000=max. CCW, 0255=max. CW)
Envía VOLUMEN	1401XXXX	FB or FA	Ídem anterior
Lee SQUELCH	1403	1403XXXX	0000 to 0255: Send/read [RF/SQL] position (RF gain level) (0000=max. CCW, 0255=11 o'clock)
Envía SQUELCH	1403XXXX	FB or FA	Ídem anterior
Lee S-METER	1502	1502XXXX	0000 to 0255: Read S-meter level (0000=S0, 0120=S9, 0240=S9+60 dB)
Envía S-METER	1502XXXX	FB or FA	Ídem anterior
Lee T	1642	1642XX	00=OFF, 01=ON
Envía T OFF	164200	FB or FA	-
Envía T ON	164201	FB or FA	-
Lee TSQL	1643	1643XX	00=OFF, 01=ON
Envía TSQL OFF	164300	FB or FA	-
Envía TSQL ON	164301	FB or FA	-
SUB BAND OFF	165900	FB or FA	-
Lee SPLIT OFFSET	1A050015	1A050015X0XX0XXX	
Lee DUPLEX OFFSET	1A050017	1A050017XXXXXX	 *10 MHz digit can be entered on only the 1200 MHz frequency band.
CI-V RTX OFF	1A05005800	FB or FA	-
Auto REPLY Off	1A05013200	FB or FA	-

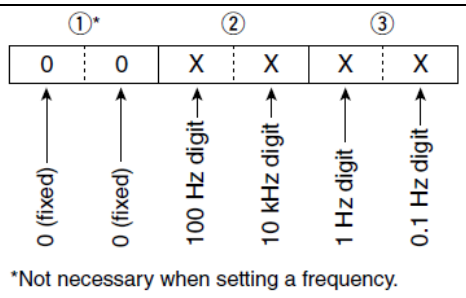
Lee T frecuencia	1B00	1B0000XXXX	
Envía T frecuencia	1B00XXXX	FB or FA	Ídem arriba (ojo *)
Lee TSQL frecuencia	1B01	1B0100XXXX	Ídem arriba
Envía TSQL frecuencia	1B01XXXX	FB or FA	Ídem arriba (ojo *)
RX (Recibe)	1C0000	FB or FA	-
TX (Emite)	1C0001	FB or FA	-

Ilustración 36: Comandos empleados

NOTA: [x] Significa que es opcional (generalmente para enviar)
NOTA: Para más información sobre los comandos ver el manual Command_IC-9100.pdf entregado con la PEC (son páginas extraídas del manual de operación).

2.7) Configuración de la IC-9100

Para el correcto funcionamiento de nuestra App se requieren configurar tres parámetros de la emisora. En el menú 60 hay que hacer coincidir los bps de la emisora con el selecionado en el módulo bluetooth HC-06 (19200 en mi caso que es lo máximo del equipo), en el 61 hay que dejar la dirección 7C y en el 62 es recomendable tenerlo a ON.

60. CI-V Baud Rate **(Default: Auto)**

Set the CI-V data transfer rate between 300, 1200, 4800, 9600, 19200 bps and "Auto."
When "Auto" is selected, the baud rate is automatically set according to the data rate of the connected controller.

61. CI-V Address **(Default: 7Ch)**

To distinguish equipment, each CI-V transceiver has its own Icom standard address in hexadecimal code.

The IC-9100's address is 7Ch.

When 2 or more IC-9100's are connected to an optional CT-17 CI-V level converter, rotate [MAIN DIAL] to select a different address for each IC-9100; the range is 01h to DFh.

62. CI-V Transceive **(Default: ON)**

Turn the transceive function using the CI-V system ON or OFF.
When this item is set to ON, changing the frequency, operating mode, etc. on the IC-9100 automatically changes those settings on other Icom transceivers or receivers, and vice versa.

Ilustración 37: Configuración menú emisora

3. Implementación

3.1) Creación de la base del proyecto desde cero

Partimos desde cero creando un proyecto nuevo con Android Studio 3.0.1 con nombre de paquete edu.uoc.android.icombluetoohcontrol (ya que la app se llama Icom Bluetooth Control) y se inicia con una Empty Activity.

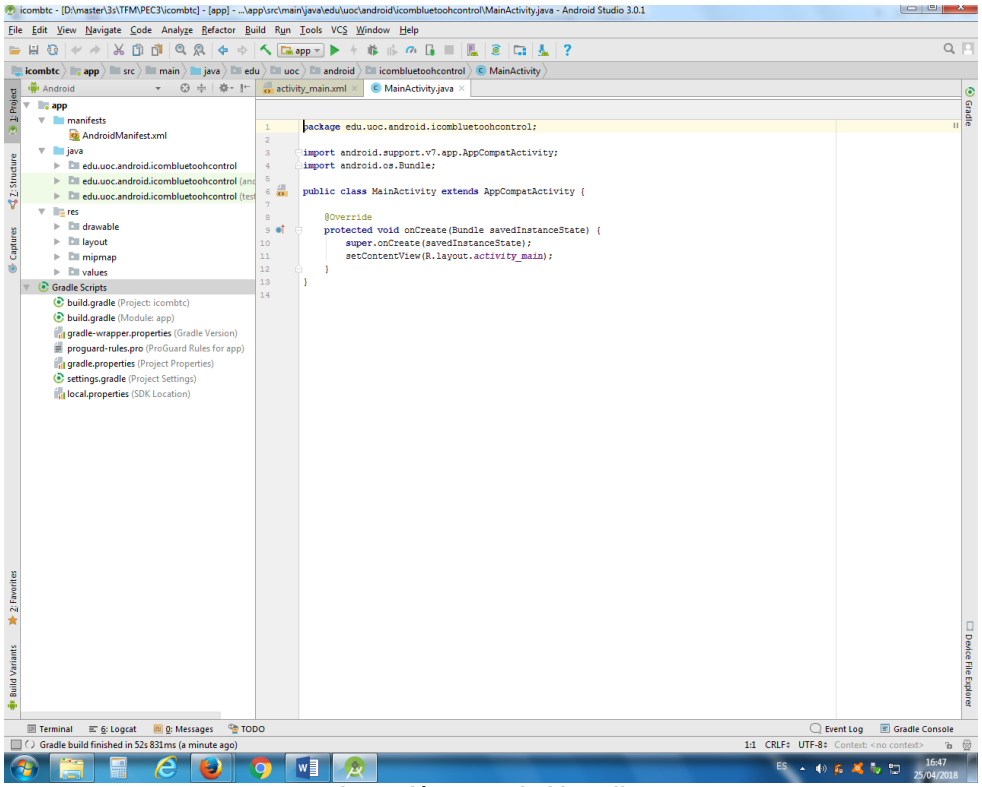


Ilustración 38: Android Studio 1

Antes que nada bloqueamos en el AndroidManifest.xml el layout a vertical solo mediante android:screenOrientation="portrait" y le ponemos un label “Icom BTC”, esto aparecerá por defecto en la bar superior y bajo el icono. Luego le ponemos el icono con una fotografía oficial de la marca ICOM. Esto es posible ya que no vamos a usar ésta versión del proyecto comercialmente, en tal caso se debería diseñar un icono propio y no usar la marca registrada.

Ilustración 39: Icono oficial

Para ello ponemos la vista en modo Android y vamos a la carpeta res y hacemos clic en new -> image -> asset y luego legacy only launcher icon.

Entonces escogemos image y buscamos el path del archivo original y luego a next, ya tenemos nuevo icono en cinco resoluciones distintas (la imagen original era de 300x300).

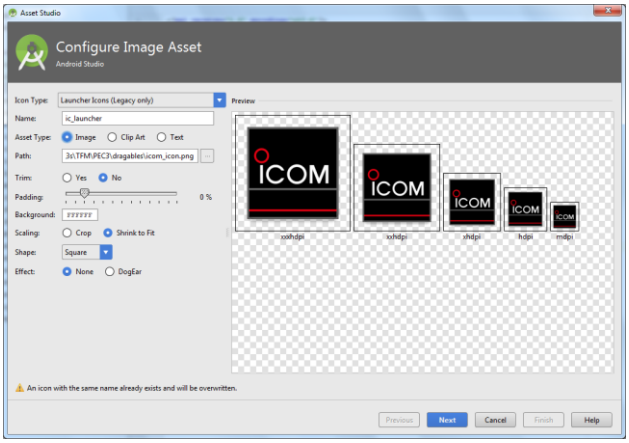


Ilustración 40: Iconos por tamaños

Vamos ahora a insertar un splash activity (pantalla de bienvenida), para ello creamos una nueva clase llamada SplashActivity.java desde la carpeta java\edu.uoc.android.icombluetoohcontrol mediante new -> java class y le ponemos el código correspondiente (ver código al final memoria, al igual que el resto de código que de mencione a partir de ahora). Luego declaramos esta actividad como principal en el manifest. Evidentemente tenemos que crear el layot activity_splash.xml desde la carpera layout mediante new -> activity -> empty activity.

Dentro de \icombtc\app\src\main\res\drawable copiamos dentro las imágenes que necesitaremos para la pantalla de bienvenida y comenzamos a editar el layout splash con dichas fotos. Queda así.

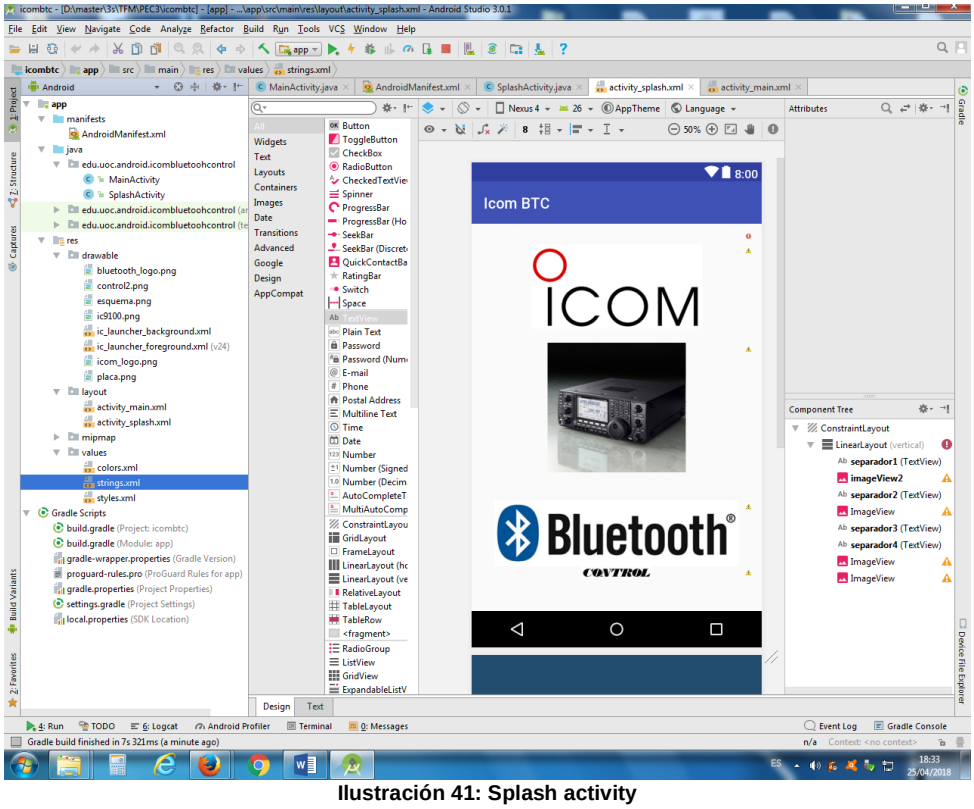


Ilustración 41: Splash activity

Probamos la App con el ADB usando un dispositivo real y capturamos la pantalla splash de bienvenida y cuando entra ya en la MainActivity.

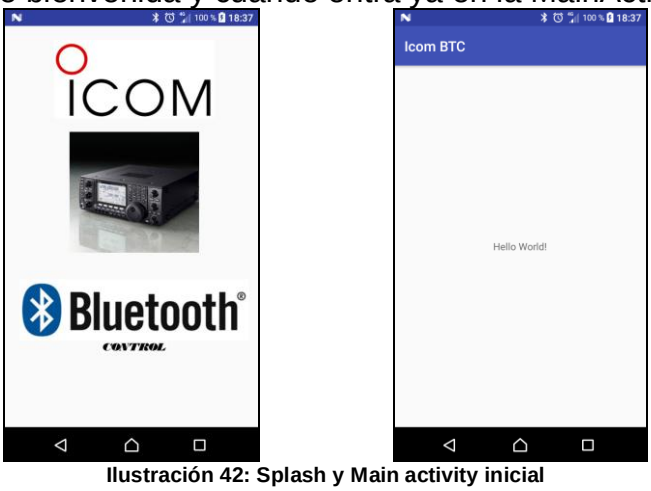


Ilustración 42: Splash y Main activity inicial

Mezclamos ahora paso a paso los tres ejemplos base previamente modificados a nuestros requerimientos funcionales, estamos hablando de añadir la función Bluetooth, la perilla de sintonía o knob y el custombar como medidor de señal s-meter, y que recuerdo que ya teníamos creados y probados por separado.

3.1.a) BLUETOOTH CONTROL

Comenzamos por darle la funcionalidad Bluetooth, para ello tendremos que editar MainActivity.java e incorporar al proyecto la clase DeviceListActivity.java, aparte de dar permisos BT en el AndroidManifest.xml, editar el layout MainActivity.xml (que se quedará como pantalla con los controles principales de control del transceptor) y crear un layout nuevo device_list.xml que nos mostrará una lista con los dispositivos Bluetooth asociados para poder conectar y crear también el device_name.xml que será cada ítem que hay que inflar en la lista. Ponemos algunas cadenas en strings.xml, la App se ha decidido dejarla exclusivamente en inglés, el motivo de esta decisión de diseño es que no es necesario pues todos los radioaficionados emplean siempre los equipos en inglés. Ponemos en el build.gradle (app) dependencias y probamos.

Ilustración 43: Lista dispositivos Bluetooth

Como podemos observar en las anteriores imágenes tras la pantalla splash de presentación (omitimos la foto) viene la pantalla del listado de dispositivos Bluetooth asociados, luego al hacer clic sobre ICOM_IC-9100_CI-V (nombre puesto a nuestro dispositivo previamente) se carga la actividad principal, donde de momento solamente tenemos botones para poder cambiar de modo de modulación al transceptor am/fm y otros.

Para probar hemos pulsado DV (un sistema de transmisión digital propiedad de ICOM conocido como D-STAR) y vemos en el toast como ha sido exitoso el comando mandado y vemos abajo del todo los comandos recibidos de la emisora, este textview de texto no se verá al final del proyecto, de momento es para hacer pruebas.

3.1.b) S-METER O MEDIDOR DE SEÑAL RECIBIDA (CUSTOM SEEK BAR)

Vamos ahora a integrar el s-meter. Para ello en cuanto a clases tendremos que añadir al proyecto la clase ProgressItem.java que es el modelo del medidor, además requeriremos la clase CustomSeekBar.java que es el controlador, y por último escribir código en la actividad principal para usarlo.

El resultado de la inserción del s-meter queda así, teniendo en cuenta que todavía no funciona, es decir señala de forma fija, ya podemos observar el programa BT base mezclado con el s-meter en MainActivity.

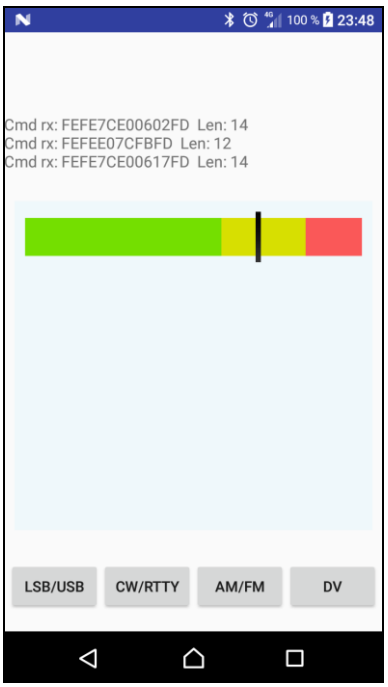


Ilustración 44: S-meter insertado

3.1.c) ROTARY KNOB CONTROL

Ilustración 45: Knob real

Para simular el dial de sintonía (como el de la foto de la emisora anterior) nos interesa un mando rotatorio y así que se parezca más nuestra app a un mando de verdad. Buscando por internet encontraremos diversos tipos de mandos del tipo volumen como los siguientes.

Ilustración 46: Knobs

Descargamos pues un ejemplo que usaremos de base para modificarlo y crear así uno a nuestro gusto, esto es que gire de forma infinita y que detecte si sube o baja para llamar a las funciones de subir/bajar frecuencia (casi todos los que existen indican sus grados absolutos sin detección de dirección, y no giran 360 grados).

Una vez conseguida la funcionalidad pretendida y probado por separado lo insertamos a nuestro proyecto, para ello añadimos al mismo las clases RoundKnowButton.java y la clase Singleton.java (requerido por la anterior).

Luego tendremos que poner en drawable las tres imágenes que componen el mando rotatorio, rotoron.png, rotoroff.png y stator.png, la estrategia de funcionamiento es tener el stator como imagen fija y hacer rotar el rotor mediante código java, al tenerlo con led verde y rojo lo aprovecharemos para poder bloquear el dial haciendo clic sobre él, de esta forma de manera alternada se verá el led verde o el rojo.

Ilustración 47: Imágenes de la perilla

El resultado tras la insercción de la perilla giratoria, además de añadir un TextView para la frecuencia y un SeekBar (todavía sin funcionalidad) es:

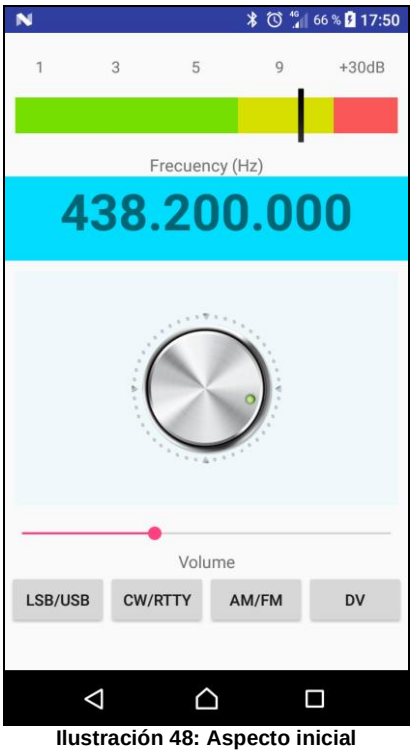


Ilustración 48: Aspecto inicial

Aunque se ha insertado un dial rotatorio para el control de la frecuencia, en cuanto al volumen y otros controles que añadiremos posteriormente se ha decidido emplear seekbars en lugar de mando giratorios, aunque en la emisora físicamente estos mandos también son rotatorios en nuestra app queda mas estético y funcional así.

3.1.d) Explicación del código hasta ahora.

Al mezclar los tres códigos fuente que ya teníamos preparados y probados por separado no hemos hablado sobre los mismos en profundidad, vamos ahora con las partes más interesantes o importantes del código java.

3.1.d.1 Conexión Bluetooth y envío/recepción de comandos

En cuanto a la conexión Bluetooth usamos la clase externa DeviceListActivity para obtener el adaptador Bluetooth, mostrar a modo de lista los dispositivos previamente vinculados, y si hacemos clic sobre algunos de ellos es llamado MainActivity mandando la MAC del dispositivo vía intent. De esta manera y gracias a la subclase connectBt nuestro programa principal por medio de dicha MAC podrá conectar con el mismo creando previamente un socket, durante este proceso mostrará un progress circular ya que extiende AsyncTask.

Seguidamente mediante el siguiente código de preparan un input y output stream para las cadenas que se recibirán o se enviarán respectivamente por el BT. Para todo ello se ha creado otra subclase llamada ConnectedThread que

extiende la clase Thread y que dispone de los siguientes métodos para gestionar la comunicación.

En el constructor obtenemos los stream.

```
private ConnectedThread(BluetoothSocket socket)
    InputStream tmpIn = null;
    OutputStream tmpOut = null;

    try {
        tmpIn = socket.getInputStream();
        tmpOut = socket.getOutputStream();
    } catch (IOException e) {
        Log.e("TAG", "IOException ConnectedThread");
    }
    mmInStream = tmpIn;
    mmOutStream = tmpOut;
}
```

Para poder mandar comandos se ha preparado el método write() que compone el comando añadiendo al parámetro recibido la cabecera FEFE y el final FD.

```
private void write(byte[] comando) {
    try {
            //comando_a_enviar = cabecera + comando + final
            byte[] both = ArrayUtils.addAll(cabeza, comando);
            byte[] three = ArrayUtils.addAll(both, fin);
            //long min comando válido 6 (en hex)
            if ((three.length >= 6)) {
                mmOutStream.write(three); //write bytes BT via outstream

                StringBuilder txCmdStr = new StringBuilder();
                for (byte b : three) {
                    txCmdStr.append(String.format("%02X", b));
                }
                Log.e("TAG", "Comando ci-v TX: " + txCmdStr + "  Len: " +
                        txCmdStr.length());
            } else {
                Log.e("TAG", "Cadena no transmitida: " + "NO VALIDA");
            }

    } catch (IOException e) {
            Toast.makeText(getBaseContext(), "write(byte[] comando): DEVICE " +
                    "UNAVAILABLE",Toast.LENGTH_SHORT).show();
            Log.e("TAG", "write(byte[] comando): DEVICE UNAVAILABLE");
            finish(); //regresa a DeviceList
    }
}
```

En cuanto a la recepción de comandos se ha creado un bucle infinito que va extrayendo los datos recibidos del input stream y mandándolos a un Handler fuera de esta subclase para gestionarlos.

```
public void run() {
    byte[] buffer = new byte[128];
    int bytes;
    // Keep looping to listen for received messages
    while (true) {
        if (btConnected) {
            try {
                bytes = mmInStream.read(buffer); //read bytes from input buffer
                String readMessage = new String(buffer, 0, bytes,
                        Charset.forName("ISO-8859-1"));
                // Send the obtained bytes to the UI Activity via handler
                mHandler.obtainMessage(MESSAGE_READ, bytes, -1,
                        readMessage).sendToTarget();
            } catch (IOException e) {
                btConnected = false;
                Log.e("TAG", "IOException mHandler ConnectedThread");
                break;
```

```
            }
        }
    } //fin while
}
```

Dicho Handler es el corazón de la recepción de comandos, tarea mucho más elaborada que el envío de los mismos, le hemos llamado rxHandler y se ha creado también como subclase.

Como podemos observar seguidamente si recibe algún dato –podría ser un comando completo o no- lo concierte en una cadena que representa su valor en hexadecimal mediante stringToHex() y luego lo manda a analizarSecuencia() que es un método que retorna si dentro de dicha secuencia existe un comando válido y su posición dentro de dicha cadena, en caso de existir un comando válido lo manda a gestionarUI() que es el encargado de mostrar en la pantalla la información recibida. Por si dentro de la cadena existe más de un comando una vez gestionado el primero se recorta y se manda de nuevo a analizarSecuencia() saliendo del bucle while cuando no existen comandos.

```
@Override
public void handleMessage(Message msg) {
    if (msg.what == MESSAGE_READ) {              //if message is what we want
        String readMessage = msg.obj.toString();

        if (msg.arg1 > 0) {                      //msg.arg1 = bytes from connect thread
            recDataString.append(readMessage);   //va acumulando lo recibido
            String recDataHex = stringToHex(recDataString.toString()); //paso a hex

            int[] infoCmd = {-1, -1, -1};
            infoCmd = analizarSecuencia(recDataHex);

            if (infoCmd[0] == 1) { //posible cmd válido, usar y borrar
                while (infoCmd[0] == 1) { //mientras exista comando
                    String cmd = recDataHex.substring(infoCmd[1], infoCmd[2]);//all cmd

                    if (cmd.substring(6, 8).equals(dirStr)) { // si lee dir 7C
                       Log.e("TAG", "Comando ci-v RX: " + cmd + "  Cabecera: " +
                                                infoCmd[1] + "  Final: " + infoCmd[2]);
                       gestionarUI(cmd); //llamo a realizar cambios en la pantalla
                    }
                    //elimina solo hasta el comando gestionado (len string is hex * 2)
                    recDataString.delete(0, (cmd.length() * 2) + (infoCmd[1] * 2));

                    //se queda con el resto no gestionado
                    if (recDataString.length() > 0) { // si aún quedan carácteres
                        recDataHex = recDataHex.substring(infoCmd[2] + 1, cmd.length());
                    } else {
                        recDataHex = "";
                    }

                    infoCmd = analizarSecuencia(recDataHex);
                } //fin while
            }
        }
    }
} //fin de handleMessage
```

Analizamos ahora el método analizarSecuencia(), como se puede observar retorna un array con la información de si hay un comando en la cadena recibida (ya que tiene que comenzar por FEFE y terminar por FD) y su posición de inicio y fin.

```
static private int[] analizarSecuencia(String cad) {
    int[] aux = {-1, -1, -1};
```

```
    int cabezacmd = cad.indexOf("FEFE"); //busca cabecera comando (podría tener echo)
    int finalcmd = cad.indexOf("FD");    //busca primer final (pueden haber dos)

    //si encuentra una cabecera y final
    if ((cabezacmd >= 0) && (finalcmd >= 0)){ //encontrado ini-fin
        if ((finalcmd - cabezacmd) >= 10) {   //encontrado posible comando válido o echo
            aux[0] = 1;
            aux[1] = cabezacmd;    //se queda con la posición
            aux[2] = finalcmd + 2; //+2 por el último FD
        }
    }

    return (aux); // retorno {si_existe_comando, cabeza, fin}
}
```

Cabe señalar que los datos recibidos son en hexadecimal, por tanto si los mostráramos en el log aparecerían caracteres extraños. Para solventar tal problema se ha creado el método stringToHex() que recibe la cadena recibida por el Bluetooth y retorna la misma representada en un string.

```
static private String stringToHex(String string) {
    StringBuilder newString = new StringBuilder();
    for (int i = 0; i < string.length(); i++) {
        newString.append(String.format("%02X", (byte) (string.charAt(i))));
        //newString.append(String.format("%02X", string.charAt(i)& 0xFF)); //ok
    }
    return newString.toString();
}
```

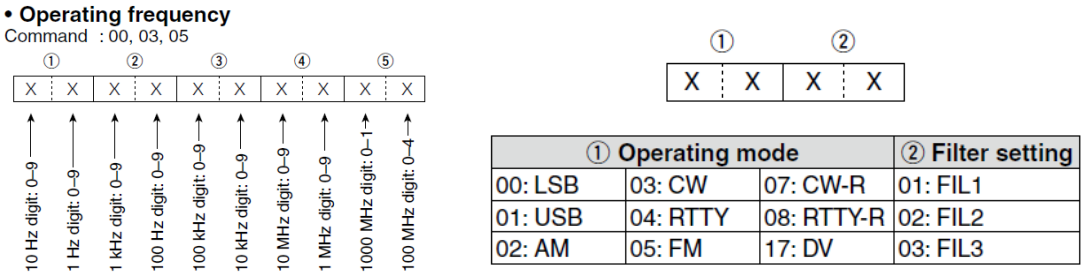
Cabe señalar que además de este último método propio para el manejo de comandos en hexadecimal también usamos en otros puntos una clase externa, la clase Hex de Apache.

Para terminar este punto falta estudiar parte del código de gestionarUI(), método que recibe como parámetro un comando válido recibido por el BT y que se encarga de gestionar la interface de usuario para mostrar la información…

```
static private void gestionarUI(String cad) {
    int lenCad = cad.length();

    String cmd = cad.substring(8, 10);
    String subCmd = cad.substring(10, 12);
    Log.e("TAG", "RX: Cmd: " + cmd + " subCmd " + subCmd + " len: " + lenCad);

    switch (cmd) {
        case "03": //frec
            if (lenCad == 22) {
                frecString.setText(formatFrec(cad.substring(10, 20)));
            } else {
                frecString.setText("blank");
            }
            break;
        case "04": //mode
            if (lenCad == 16) {
                infoString.setText(formatModo(cad.substring(10, 14)));
            } else {
                infoString.setText("??");
            }
            break;
       case "15": //S-meter
            if (lenCad == 18) {
                if (subCmd.equals("02"))
                    mCSeekbar.setProgress((int) (Double.parseDouble
                            (cad.substring(12, 16)) / 2.56));
            }
            break;

          **********
```

Como podemos observar si recibe el comando 03h formatea la frecuencia y la manda al TextView que muestra la frecuencia, y si recibe el comando 04h formatea el modo de modulación y lo manda a su TextView correspondiente, y así con el resto de los posibles casos iremos ampliando el switch.

El formateo de los datos normalmente es necesario ya que o bien no se reciben los datos en el orden real o bien hay que traducirlos, lo vemos a continuación:



① Operating mode			② Filter setting
00: LSB	03: CW	07: CW-R	01: FIL1
01: USB	04: RTTY	08: RTTY-R	02: FIL2
02: AM	05: FM	17: DV	03: FIL3

Ilustración 49: Protocolo CI-V 4

3.1.d.2 El S-meter o medidor de señal (customseekbar)

Se ha insertado al proyecto la clase CustomSeekBar que gestiona tal elemento, su base de funcionamiento es el método onDraw() que dibuja el mismo.

```
protected void onDraw(Canvas canvas) {
    if (mProgressItemsList.size() > 0) {
        int progressBarWidth = getWidth();
        int progressBarHeight = getHeight();
        int thumboffset = getThumbOffset();
        int lastProgressX = 0;
        int progressItemWidth, progressItemRight;
        for (int i = 0; i < mProgressItemsList.size(); i++) {
            ProgressItem progressItem = mProgressItemsList.get(i);
            Paint progressPaint = new Paint();
            progressPaint.setColor(getResources().getColor(
                    progressItem.color));

            progressItemWidth = (int) (progressItem.progressItemPercentage
                    * progressBarWidth / 100);

            progressItemRight = lastProgressX + progressItemWidth;

            // for last item give right to progress item to the width
            if (i == mProgressItemsList.size() - 1
                    && progressItemRight != progressBarWidth) {
                progressItemRight = progressBarWidth;
            }
            Rect progressRect = new Rect();
            progressRect.set(lastProgressX, thumboffset / 2,
                    progressItemRight, progressBarHeight - thumboffset / 2);
            canvas.drawRect(progressRect, progressPaint);
            lastProgressX = progressItemRight; //sumar número pone separación colores
        }
        super.onDraw(canvas);
    }

}
```

Desde MainActivity lo configuramos a nuestros requerimientos.

```
private void initDataToCSeekbar() {
    float totalSpan = 1200;
    float greenSpan = 700;
    float yellowSpan = 300;
    float redSpan = 200;
```

```
    ArrayList<ProgressItem> progressItemList;
    ProgressItem mProgressItem;

    progressItemList = new ArrayList<ProgressItem>();

    // red span
    mProgressItem = new ProgressItem();
    mProgressItem.progressItemPercentage = ((greenSpan / totalSpan) * 100);
    Log.i(TAG, "Smeter: " + mProgressItem.progressItemPercentage);
    mProgressItem.color = R.color.green;
    progressItemList.add(mProgressItem);
    // green span
    mProgressItem = new ProgressItem();
    mProgressItem.progressItemPercentage = (yellowSpan / totalSpan) * 100;
    mProgressItem.color = R.color.yellow;
    progressItemList.add(mProgressItem);
    // yellow span
    mProgressItem = new ProgressItem();
    mProgressItem.progressItemPercentage = (redSpan / totalSpan) * 100;
    mProgressItem.color = R.color.red;
    progressItemList.add(mProgressItem);
    // ---
    mCSeekbar.initData(progressItemList);
    mCSeekbar.invalidate();
    mCSeekbar.setEnabled(false); //deshabilita desplazamiento táctil
}
```

3.1.d.3 El dial rotatorio (knob)

La perilla de sintonía aporta realismo a este proyecto, simulando a la original de la emisora, para ello se ha insertado la clase RoundKnobButton. Básicamente se trata de hacer rotar una imagen según la detección de gestos, al mismo tiempo admite el evento clic para el bloqueo del mismo.

Siendo ivRotor un Bitmap la imagen inicial se selecciona mediante:

```
ivRotor.setImageBitmap(state?bmpRotorOn:bmpRotorOff); //dial bloq rojo al iniciar
```

Y el detector de gestos es:

```
gestureDetector = new GestureDetector(getContext(), this);
```

La rotación se consigue mediante:

```
public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float
distanceY) {
   if (!dialBloqueado) { // Si el dial no está bloqueado ruedo y cuento
      float x = e2.getX() / ((float) getWidth());
      float y = e2.getY() / ((float) getHeight());
      float rotDegrees = cartesianToPolar(1 - x, 1 - y);// 1- to correct axis direction

      if (!Float.isNaN(rotDegrees)) {
         // rotate our imageview
         setRotorPosAngle(rotDegrees);
         // get a linear scale
         float scaleDegrees = rotDegrees + 150; // given parameters, we go from 0 to 300
         // get position percent
         int percent = (int) (scaleDegrees / 3);// calcula el % a mostrar
         if (m_listener != null) m_listener.onRotate(percent);
         return true; // consumed
      } else {
         return false; // not consumed
      }
   }else {
      return false; // not consumed
   }
}
```

Del dial tan solo sabemos el porcentaje, por tanto en la actividad principal para detectar si la perilla va hacia arriba o hacia abajo comparamos el valor de dicho porcentaje.

```
if (percentage2 - percentage != 0) {
    if (((percentage2-percentage < 0) && (percentage2 - percentage > -18))
            || (percentage2 - percentage > 100)) {
        cambiarFrecuencia("SUBIR");
    } else {
        cambiarFrecuencia("BAJAR");
    }
    percentage2 = percentage; //memorizo % anterior
}
```

3.2) Ampliación de funcionalidades

Ya tenemos creada la base del proyecto, pudiendo incluso conectar con la emisora y usar los cuatro botones de cambio de modo como el AM/FM para probar la comunicación Bluetooth.

Sin embargo no funcionan todavía ni la pantalla de información, ni el dial, ni el s-meter, amén de requerir el proyecto la ampliación de funcionalidades.

3.2.a) Actualizador automático

Para que funcione el s-meter, se visualice la frecuencia y el modo etc, necesitamos un sistema que lance una función cada cierto tiempo de forma repetida y sin parar para que haga tal trabajo, lo que hará es mandar el comando de lectura correspondiente y gestionar la respuesta, el motivo es bien claro, si por ejemplo alguien cambia manualmente en el equipo alguno de estos parámetros tiene que quedar reflejado en la pantalla de la app, por tanto se tiene que actualizar prácticamente cada segundo.

Para tal fin estudiaremos como hacer en java una especie de setInterval() que tiene el famoso lenguaje web javascript. Sencillamente ponemos en el onCreate el siguiente código y probamos en el log si se lanza cada segundo.

```
new Timer().scheduleAtFixedRate(new TimerTask(){
  @Override
  public void run(){
    Log.e("tag", "Actualización");
  }
},0,1000);
```

Una vez probamos añadimos código, el cual ampliaremos más adelante.

```
new Timer().scheduleAtFixedRate(new TimerTask(){
    @Override
    public void run(){
      if(btConnected) {
        switch (turno) {
            case 1:
                byte[] frecComando = {(byte) 0x03};                 //frec => 03
                mConnectedThread.write(frecComando);
                turno = 2;
                break;
            case 2:
                byte[] modeComando = {(byte) 0x04};                 //mode => 04
                mConnectedThread.write(modeComando);
                turno = 1;
                break;
```

```
            }
        }
    }
},4000,400);
```

Podemos observar cómo tras 4 segundos iniciales de espera cada 400 milisegundos se lanzará un comando distinto para actualizar la app, de momento se piden los datos sobre la frecuencia y el modo de modulación (comandos 03h y 04h respectivamente). Recordamos que es el método write() el que añade luego la cabecera y el fin para enviar el comando completo.

3.2.b) Mejora del layout principal

Vamos ahora a mejorar el aspecto visual y a insertar más controles, al final dejaremos la app en negro de fondo, el motivo de tal decisión de diseño es que se ven mejor los controles especialmente el dial knob.

Vamos a dejar la App con fondo negro ya que tras probar y preguntar a potenciales usuarios llegan al acuerdo de que se ve mejor así, se han añadido botones (muchos tendrán doble función mediante el efecto dblClick, y algunos admitirán también la pulsación larga), se han puesto dos SeekBar uno para el volumen y otro para el silenciador squelch. En la pantalla de frecuencia se han añadido textViews para mostrar información relevante como el modo de modulación y otros.

3.2.c) Layouts multi pantalla

Como ya sabemos uno de los problemas fundamentales de Android es la fragmentación, tanto en cuanto a versiones se refiere como en tamaños de pantalla. Preocupándonos ahora de este segundo punto vamos a modificar el layout para que funcione perfectamente con pantallas entre 4 y 10 pulgadas.

Tras varias pruebas para ello modificaremos algunos aspectos y lo guardaremos en cuatro archivos distintos con el mismo nombre, uno para pantallas menores o iguales a 5” (activity_main-sw320dp), otro para las de 5.5 y 6” (activity_main-sw360dp), otro para las de 7” (activity_main-sw600dp) y otro para las de 10 o 10,1” (activity_main-sw720dp). Los resultados son:

Teléfono de 4” (LG 50)

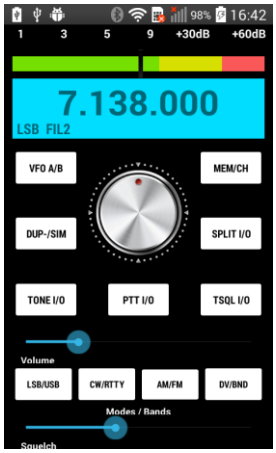


Ilustración 50: Aspecto en 4"

Teléfono de 5” (Alcatel One Touch)

Ilustración 51: Aspecto en 5"

Teléfono de 5,5” (Sony L1)

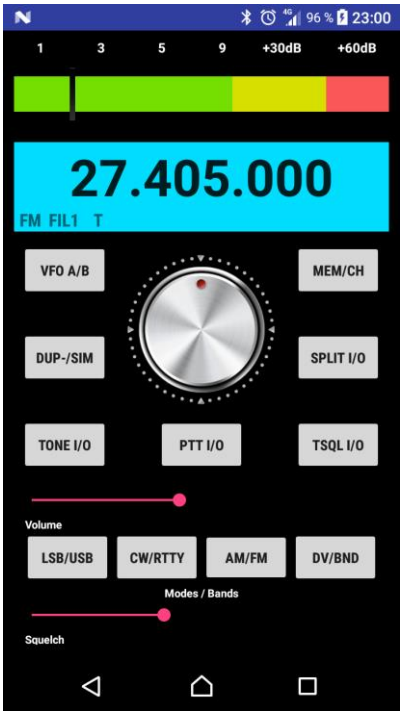


Ilustración 52: Aspecto en 5.5"

Tablet de 7” (Lenovo Tab 4)

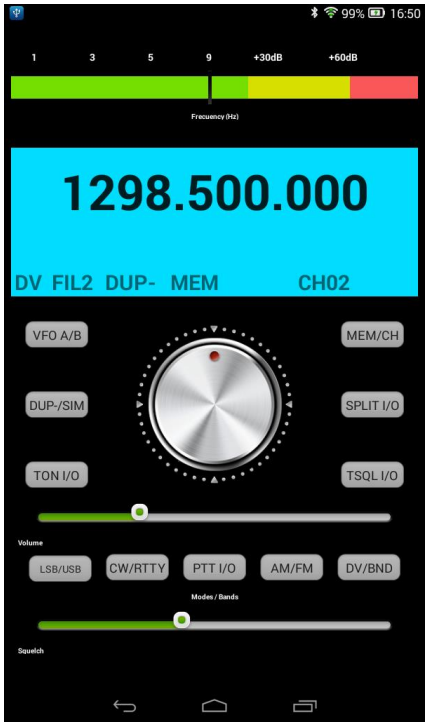


Ilustración 53: Aspecto en 7"

Tablet de 10,1” (Samsung Galaxy Tab A6)

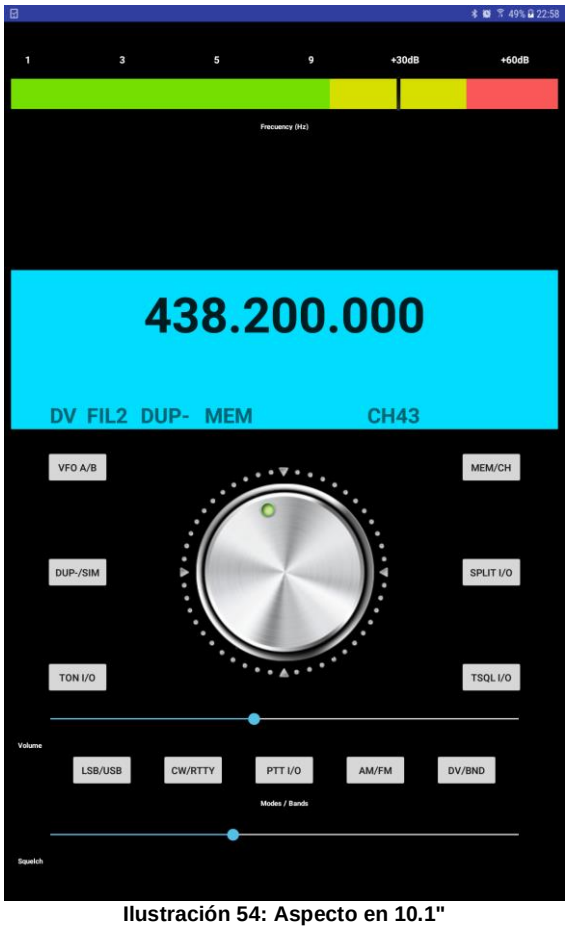


Ilustración 54: Aspecto en 10.1"

3.2.d) Botones

Necesitamos ahora crear código que gestione los botones que acabamos de insertar en la UI, los deseamos todos con el efecto doble clic, lo cual conseguiremos mediante un Handler temporizado, luego se trata de según el tipo de evento mandar un comando y otro. Algunos botones (CW/RTTY, DUP-/SIM y SPLIT I/O) tienen también el evento de pulsación larga, por lo que tienen triple funcionalidad. Vemos ahora parte del código.

```
@Override
public void onClick(View v) { //gestión integral de eventos de botones (incluye dblClick)

   Handler clickHandler = new Handler();

   switch (v.getId()) {
      case R.id.buttonLsbUsb:
         dblClick++;
         clickHandler.postDelayed(new Runnable() {
            @Override
            public void run() { //lsb-usb
               if (dblClick == 1) {       //lsb =>  0060
                  byte[] lsbComando = {(byte) 0x06, (byte) 0x00};
                  if(btConnected) mConnectedThread.write(lsbComando);
               } else if (dblClick == 2) { //usb => 0601
                  byte[] usbComando = {(byte) 0x06, (byte) 0x01};
                  if(btConnected) mConnectedThread.write(usbComando);
               }
               dblClick = 0;
            }
         }, 500);
         break;
      case R.id.buttonCwRtty:
         dblClick++;
         clickHandler.postDelayed(new Runnable() {
            @Override
            public void run() { //cw-rtty
               if (dblClick == 1) {       //cw => 0603
                  byte[] cwComando = {(byte) 0x06, (byte) 0x03};
                  if(btConnected) mConnectedThread.write(cwComando);
               } else if (dblClick == 2) { //rtty => 0604
                  byte[] rttyComando = {(byte) 0x06, (byte) 0x04};
                  if(btConnected) mConnectedThread.write(rttyComando);
               }
               dblClick = 0;
            }
         }, 500);
         break;
      case R.id.buttonAmFm:
         dblClick++;
         clickHandler.postDelayed(new Runnable() {
            @Override
            public void run() { //am-fm
               if (dblClick == 1) {       //am => 0602
                  byte[] amComando = {(byte) 0x06, (byte) 0x02};
                  if(btConnected) mConnectedThread.write(amComando);
               } else if (dblClick == 2) { //fm => 0605
                  byte[] fmComando = {(byte) 0x06, (byte) 0x05};
                  if(btConnected) mConnectedThread.write(fmComando);
               }
               dblClick = 0;
            }
         }, 500);
         break;
            }
}
```

3.2.e) Salvapantallas

Seguidamente se decide quitar el salvapantallas mientras se use la app, para ello le daremos permiso en el manifest mediante…
<uses-permission android:name="android.permission.WAKE_LOCK" />
y luego en el onCreate de MainActivity ejecutaremos…
getWindow().addFlags(LayoutParams.FLAG_KEEP_SCREEN_ON);

3.2.f) Botón MEM/CH

Con el clic de este botón se pretende poner la emisora en modo memorias, y con el doble clic se pretende mandar al equipo el número de memoria deseada (de 01 a 99, el equipo dispone de 99 por banda). Ponemos el mismo código para el doble clic visto antes y se añade la siguiente función que crea un dialog con radios para seleccionar el canal.

```
protected void showSpinnerCh(){
   AlertDialog.Builder b = new AlertDialog.Builder(this);

   String[] options = new String[99];
   String relleno="";
   for (Integer i=0; i<99; i++){
      if (i < 9) {
         relleno = "0" + (i+1);
         options[i]="CH " + relleno;
      }else{
         options[i]="CH " + (i+1);
      }

   }

   b.setTitle("Select memory channel" );
   b.setIcon(R.mipmap.ic_launcher);
   b.setSingleChoiceItems(options, -1, new DialogInterface.OnClickListener(){
      public void onClick(DialogInterface dialog, int which)
      {
         String ch = Integer.toString(which + 1 );
         if (ch.length() < 2) ch = "0" + ch;
         String chComando = "08" + "00" + ch; //comando en String
         if(btConnected) mConnectedThread.write(decodeHex(chComando));    //lo paso a hex
         dialog.dismiss();
      }

   });
   b.setPositiveButton("CANCEL", new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int id) {
         dialog.dismiss();
      }
   });
   b.show();
}
```

El comando mandado es 0800xx (+cabeza y fin) donde xx es el canal de 01 a 99.

Vemos ahora el aspecto tras el doble clic en el botón:

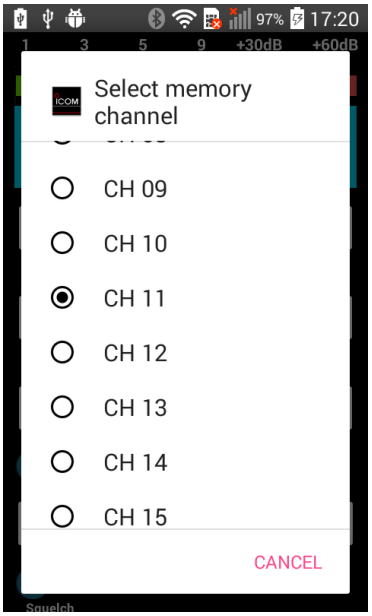


Ilustración 55: Menú seleccionar canal

3.2.g) Menú o panel lateral (Navigation Drawer)

Uso el de Mike Penz, para ello ponemos en el gradle.app las dependencias con compile.

```
compile("com.mikepenz:materialdrawer:6.0.2@aar") { //navigation drawer
   transitive = true
}
compile 'com.android.support:recyclerview-v7:26.1.0'
compile 'com.android.support:design:26.1.0'
compile 'com.android.support:support-annotations:26.1.0'
```

Y luego añadimos en el onCreate new DrawerBuilder().withActivity(this).build(); y con eso ya se ve el menú lateral.

Luego ampliamos código…

```
AccountHeader headerResult = new AccountHeaderBuilder()
        .withActivity(this)
        .withHeaderBackground(R.drawable.ic9100) //header con foto
        .addProfiles(
                new ProfileDrawerItem()
                        .withName("IC-9100 BLUETOOTH CONTROL")

.withIcon(getResources().getDrawable(R.drawable.icom_icono_round))
        )
        .withOnAccountHeaderListener(new AccountHeader.OnAccountHeaderListener() {
            @Override
            public boolean onProfileChanged(View view, IProfile profile, boolean
currentProfile) {
                return false;
            }
        }).build();
PrimaryDrawerItem item1 = new PrimaryDrawerItem().withIdentifier(1).withName(
        "Select Software TS").withIcon(R.drawable.ic_shuffle);
PrimaryDrawerItem item2 = new PrimaryDrawerItem().withIdentifier(2).withName(
        "Extra Controls").withIcon(R.drawable.ic_controls);
PrimaryDrawerItem item4 = new PrimaryDrawerItem().withIdentifier(4).withName(
        "GPS").withIcon(R.drawable.ic_mylocation);
PrimaryDrawerItem item6 = new PrimaryDrawerItem().withIdentifier(6).withName(
        "Help").withIcon(R.drawable.ic_help);
PrimaryDrawerItem item7 = new PrimaryDrawerItem().withIdentifier(7).withName(
```

```
        "Connect/Disconnect").withIcon(R.drawable.ic_bluetooth);
PrimaryDrawerItem item8 = new PrimaryDrawerItem().withIdentifier(8).withName(
        "Exit").withIcon(R.drawable.ic_exitapp);
mDrawer = new DrawerBuilder()
        .withActivity(this)
        .withAccountHeader(headerResult)
        .addDrawerItems(
                item1,item2,item4,
                new DividerDrawerItem(),
                item6,
                new DividerDrawerItem(),
                item7,item8
        ).withOnDrawerItemClickListener(new Drawer.OnDrawerItemClickListener() {
            @Override
            public boolean onItemClick(View view, int position, IDrawerItem drawerItem)
            {
                // do something with the clicked item :D
                if (position == 1) showSpinnerTS();
                else if (position == 2)showDialogExtraC();
                else if (position == 3) showDialogGPS();
                else if (position == 5) showHelp();
                else if (position == 7) finish(); //layout anterior
                else if (position == 8) finishAffinity(); //requiere API min 16
                return true;
            }
        }).build();
```

Y nos queda así (ya solo queda programar las opciones pertinentes):

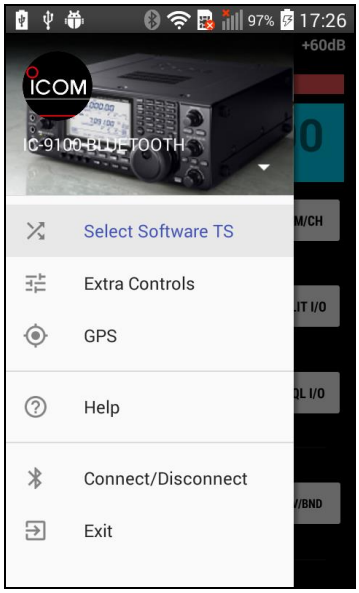


Ilustración 56: Menú lateral

3.2.h) Incorporar al proyecto la clase HEX de Apache

Este archivo Hex.java nos va a facilitar el trabajo con hexadecimales especialmente en las conversiones de tipo, hasta ahora hemos usado alguna propia pero como necesitamos más es mejor usar la de Apache.

Comenzaremos añadiendo al gradle.app

compile 'commons-codec:commons-codec:1.11'

Seguidamente ponemos en el proyecto el archivo Hex.java y le asignamos el nombre de nuestro paquete. Luego lo importamos en Main mediante…
import static edu.uoc.android.icombluetoohcontrol.Hex.decodeHex;

//clase HEX de apache

Y para usarlos (la clase obliga a usar excepciones)…

```
byte[] vComando = new byte[]{};
try{vComando = decodeHex(volComando);} catch (DecoderException de){}
if(btConnected) mConnectedThread.write(vComando); //lo mando en hex de byte[]
```

3.2.i) Dialog para entrar la frecuencia directamente por teclado

Vamos ahora a poner un dialog dinámico para poder entrar la frecuencia directamente mediante teclado, la idea es que aparezca al hacer clic sobre el TextView que muestra la frecuencia, y una vez introducida la misma al hacer clic en el ok se tiene que mandar a la emisora el comando.

Creamos un layout llamado input_dialog-xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
android:padding="10dp">
    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Enter new frecuency please"
    android:id="@+id/textView"
    />
    <EditText
        android:id="@+id/edittext"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="x{4}.xxx or xxxxxxxxxx"
        android:inputType="numberDecimal"
        android:maxLength="10"
        android:padding="10dp" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:text="Examples:"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:text="7.100"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:text="145.700"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:text="1240500000"
        android:textAppearance="?android:attr/textAppearanceMedium" />
</LinearLayout>
```

Seguidamente le habilitamos el evento onClick al textView e introducimos dentro el código…

```
        alertDialogBuilder.setCancelable(false)
                .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                        String cad = editText.getText().toString();
                        int len = cad.length();
                        if (((len == 5 || len == 6 || len == 7 || len == 8)
                                && cad.substring(len-4,len-3).equals("."))     //x{4}.xxx
                                ||((len == 10) && (cad.indexOf(".") == -1))){ //xxxxxxxxxx
                            if(len != 10)
                                cad=cad.substring(0,len-4)+cad.substring(len-3,len); //quita.
                            if (len == 5) {
                                cad = "000" + cad + "000";
                            }else if (len == 6) {
                                cad = "00" + cad + "000";
                            }else if (len == 7) {
                                cad = "0" + cad + "000";
                            }else if (len == 8) {
                                cad = cad + "000";
                            }
                            cad = unformatFrec(cad);
                            cad = "05" + cad;
                            try {
                                byte[] nFrecuency = decodeHex(cad);
                                if (btConnected && !bloqueado)
mConnectedThread.write(nFrecuency);
                            } catch (DecoderException de) {
                                Log.e("TAG", "Exception: " + "de " + "tecladoFrec");
                            }
                            turno = 1;       //para que lea frecuencia primero
                            onSend = false; //activa automático
                        }else{
                            editText.setError("format mhz.xxx or xxxxxxxxxx");
                            onSend = false; //activa automático
                        }
                    }
                })
                .setNegativeButton("Cancel",
                    new DialogInterface.OnClickListener() {
                       public void onClick(DialogInterface dialog, int id) {
                          onSend = false;
                          dialog.cancel();
                       }
                    });
              AlertDialog alert = alertDialogBuilder.create();
              alert.show();
        }
});
```

Como se aprecia y debido al protocolo de la emisora ICOM durante el proyecto se han tenido que usar conversiones a y de hexadecimal además de alguna función propia para adaptar la secuencia (en este caso unFormatFrec() y decodeHex()). Vemos el resultado:

Ilustración 57: Menú frecuencia directa

3.2.j) Dial perilla knob rotativo

Ahora que ya podemos cambiar la frecuencia de forma directa por teclado vamos a comenzar a trabajar con la perilla o dial rotativo (Knob). Este mando ya lo teníamos insertado y funcionaba el movimiento up/down. Lo que hacemos ahora es llamar a un método cuando lo rodemos para que suba o baje la frecuencia.

```
private void cambiarFrecuencia(String subbaj){
    esperarAuto(1000);

    sumador++;
    if (sumador%5==0) {//variable de sensibilidad del dial
       String fAnt = frecString.getText().toString(); //frec anterior de pantalla
       if (!fAnt.equals("blank")&&!fAnt.equals("not
connected")&&!fAnt.equals("0000.000.000")
              && !fAnt.equals("connected")){
            fAnt = formatAx(fAnt, 12); //rellena ceros izq
            //quita puntos
            fAnt = (fAnt.substring(0, 4)+fAnt.substring(5, 8) + fAnt.substring(9, 12));
            int fNum = Integer.parseInt(fAnt);          //String -> Int

            if (!tsChanged) mTS = tsDefecto(fNum); //calcula ts defecto

            int fAux = frecRedondeada(fNum, subbaj);
            if (frecValida(fAux)) fNum = fAux;      //aumenta ts

            String fPos = Integer.toString(fNum);  // Int -> String (frec. posterior)

            //pone comas de nuevo (frec. posterior con puntos)
            String fPosPunt="";
            if (fPos.length() < 7){
                fPosPunt = formatAx(fPos, 7);} //rellena ceros izq hasta 7
            else{
                fPosPunt = fPos;
            }

            int len = fPosPunt.length();
            fPosPunt = fPosPunt.substring(0, len - 6) + "." +   //pone puntos
                       fPosPunt.substring(len - 6, len - 3) +
                       "." + fPosPunt.substring(len - 3, len);

            frecString.setText(fPosPunt);   //visualiza nueva frec en pantalla

            fPos = formatAx(fPos, 10); //la formatea a len=10
            fPos = unformatFrec(fPos);      //la prepara para enviar
            fPos = "05" + fPos;             //le añade delante el comando 05

            try {
                byte[] newFrecuency = decodeHex(fPos); //lo decodifica como Hex
                if (btConnected && !bloqueado) mConnectedThread.write(newFrecuency);
            } catch (DecoderException de) {
                Log.e("TAG", "Exception: " + "de " + "dial changeFrec");
            }
            turno = 1; //para que auto lea frecuencia primero
        }
    }
}
```

Lo probamos y funciona perfectamente. Si bien el incremento/decremento está fijo a 1khz. Esto se tiene que arreglar poniendo un salto TS configurable, la idea es ponerlo en el menú lateral, no obstante como en el equipo real cada banda suele usar un salto TS por defecto lo que haremos es que automáticamente se escoja al cambiar de frecuencia el salto más usado habitualmente en la banda actual, pero que el usuario pueda cambiarlo en el menú lateral.

Además lo habitual en los equipos de radio es que en el primer movimiento del dial se ajusten los últimos dígitos a una frecuencia divisible por el TS.

Pondremos por defecto los siguientes saltos TS:

HF (30Khz-26Mhz) = 1000Hz
HF (26-30Mhz) = 5000Hz
VHF (30-60 y de 108-136Mhz) = 5000Hz
VHF (136-174Mhz)= 12500Hz
UHF (420-480Mhz) = 25000Hz
UHF (1240-1320Mhz) = 25000Hz

Por otro lado hay que tener en cuenta que en esta tabla podemos ver las frecuencias que podemos seleccionar en el transceptor ICOM IC-9100, por tanto cuando subamos o bajemos frecuencia con el dial cuando lleguemos a uno de los extremos sería interesante saltar al otro (es lo que hace la emisora realmente con su mando físico). Por ejemplo, si estamos subiendo en HF y llegamos a 60.000.000 tendríamos que pasar al principio 0.030.000 e igual a la inversa. Por lo tanto sería ideal analizar continuamente el número en cada cálculo. De momento lo dejamos así ya que si llegamos a un límite sencillamente no nos deja más. Pero si creamos seguidamente el salto de TS.

3.2.k) Salto TS

Comenzamos este punto ampliando las opciones del menú lateral y dándole función a la primera que es el salto TS (Tuning Step), le llamamos showSpinnerTs(), es la cantidad que se sumará o restará al subir o bajar frecuencia a la disponible en ese momento.

```
protected void showSpinnerTS() {
    AlertDialog.Builder b = new AlertDialog.Builder(this);

    String[] options = {"Default", "1 Hz","10 Hz","100 Hz","1 kHz","5 kHz","6.25 kHz","9
kHz",
                        "10 kHz","12.5 kHz","20 kHz","25 kHz","50 kHz","100 kHz","1
MHz"};

    final int[] misTS={1, 10, 100, 1000, 5000, 6250, 9000, 10000, 12500,
                       20000, 25000, 50000, 100000, 1000000};

    if (!tsChanged) {
        String fAc = frecString.getText().toString(); //obtengo frec actual de pantalla
        if (!fAc.equals("blank")&&!fAc.equals("not
connected")&&!fAc.equals("0000.000.000")
                &&!fAc.equals("connected")) {
            fAc = formatAx(fAc, 12); //rellena ceros izq
            fAc = (fAc.substring(0, 4) + fAc.substring(5, 8) + fAc.substring(9,
12));//quita ..
            int fNum = Integer.parseInt(fAc);      //String -> Int
            mTS = tsDefecto(fNum); //calcula ts defecto
        }
    }

    int elem=1;
    for(int i=0; i<misTS.length; i++){if (misTS[i] == mTS){elem = i+1;}} //TS actual

    b.setTitle("Select software TS");
    b.setIcon(R.mipmap.ic_launcher);
    b.setSingleChoiceItems(options, elem, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            if(which == 0) {
                tsChanged = false;    //chivato ts cambiado (defecto)
            }else {
```

```
                mTS = misTS[which-1]; //cambia TS
                tsChanged = true;     //chivato ts cambiado (manual)
            }

            dialog.dismiss();         //cierro diálogo
            mDrawer.closeDrawer();    //cierro menú lateral
        }

    });
    b.setPositiveButton("CANCEL", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.dismiss();
        }
    });
    b.show();
}
```

Es aspecto es el siguiente…

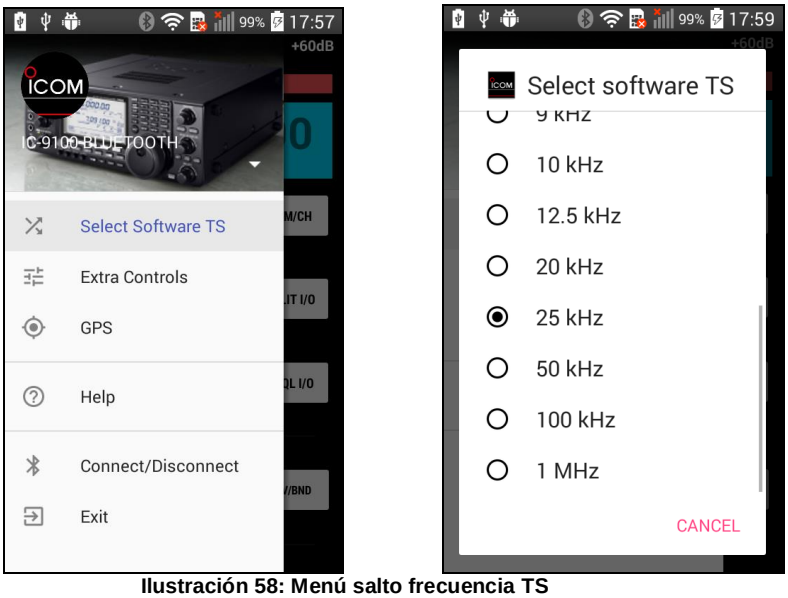


Ilustración 58: Menú salto frecuencia TS

Para darle funcionalidad creamos una variable global mTS que será cambiada al pulsar uno de éstos ítems, de esa manera esa variable la podrá usar la perilla de sintonía.

Añadimos a la vez la función tsDefecto() que retorna el salto por defecto según la frecuencia actual. El funcionamiento será que cada vez que cambiemos de frecuencia mTS cogerá el valor por defecto mediante tsDefecto(), pero si vamos a la primera opción del menú lateral la podremos cambiar nuestra voluntad. El código queda así:

```
private static int tsDefecto(int frec) {
    int result=0;
    if(frec < 26000000){ //frec < 26Mhz
        result = 1000; //ts = 1kHz
    }else if(frec < 136000000){//frec < 136Mhz
        result = 5000; //ts = 5kHz
    }else if(frec < 174000000) {//frec < 174Mhz
        result = 12500; //ts = 12,5kHz
    }else{ //frec < 1320Mhz
        result = 25000; //ts = 25kHz
    }
    return(result);
}
```

3.2.l) Cambiar de banda directamente

Ilustración 59: Teclado real bands

Las bandas de radioaficionado que cubre el equipo en cuestión y de las cuales dispone acceso directo por teclado numérico (como se ve en la anterior foto) son 1.8, 3.5, 7,10, 14, 18, 21, 24, 28 y 50 Mhz, además tiene las bandas de 144, 432 y 1200 MHz. Por otro lado sería interesante tener también acceso directo a otras bandas importantes pero que no son de radioaficionado como la radio comercial de AM, la banda PMR, banda ciudadana etc.

Para tal función habilitaremos el botón DV/BAND de manera que al pulsar dos veces se abra la posibilidad de escoger una de estas bandas, y al pulsar ir directamente al centro de la misma, cambiando de paso el modo y el salto ts.

El código de momento lo tenemos así…

```
protected void showSpinnerBand() {
    AlertDialog.Builder b = new AlertDialog.Builder(this);
    onSend = true; //apaga auto

    String[] options = {"1.8","3.5","7","10","14","18","21","24","28","50","144","432",
                        "1200", "AM RADIO", "OL", "CB", "AIR", "MARINE", "PMR"};

    String fAnt = frecString.getText().toString(); //obtengo frec anterior de pantalla
    int fNum=0;
    if (!fAnt.equals("blank")&&!fAnt.equals("not
connected")&&!fAnt.equals("0000.000.000" )
            && !fAnt.equals("connected")) {
        fAnt = formatAx(fAnt, 12); //rellena ceros izq y quita puntos
        fAnt = (fAnt.substring(0, 4) + fAnt.substring(5, 8) + fAnt.substring(9, 12));
        fNum = Integer.parseInt(fAnt);    //String -> Int (frecuencia actual en número)
    }

    int[] misBandN={2700000, 5500000, 9000000, 12000000, 16000000, //calc. banda actual
                    19500000,23000000, 26000000, 33000000, 60000000,
                    174000000, 470000000, 1320000000};
    int elem=0;
    for(int i=0; i<misBandN.length; i++ ){
        if (fNum<=misBandN[i]){
            elem=i;
            break;
        }
    }

    b.setTitle("Select band");
    b.setIcon(R.mipmap.ic_launcher);
    b.setSingleChoiceItems(options, elem, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            String[] misBand={"1910000", "3560000", "7060000", "10130000", "14200000",
                              "18130000","21300000", "24980000", "28500000", "51000000",
                              "145000000", "435000000", "1295100000", "1125000",
"252000",
```

```
                                "27205000", "124750000", "156800000", "446056250"};

            String[] misModes={"CW", "LSB", "LSB", "CW", "USB", "USB","USB", "USB",
"USB",
                                "USB", "FM", "FM", "FM", "AM", "AM", "AM", "AM", "FM",
"FM"};
            proxModo=misModes[which]; //obiene modo por defecto por banda para mandar

            int[] miDTS={1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000,
5000,
                        12500, 25000, 9000, 9000, 5000, 5000, 50000,6250};
            mTS=miDTS[which]; //obiene TS por defecto por banda
            tsChanged=true;

            String fBand = misBand[which];     //obtiene frec
            fBand = formatAx(fBand, 10); //añade ceros izq
            fBand=unformatFrec(fBand);         //la prepara para enviar
            fBand="05"+fBand;                  //le añade delante el comando 05

            try {
                byte[] newFrecuency = decodeHex(fBand); //lo decodifica como Hex
                if (btConnected && !bloqueado) mConnectedThread.write(newFrecuency);
            } catch (DecoderException de) {
                Log.e("TAG", "Exception: " + "de " + "Band select");
            }

            turno = 99;             //cambia primero el modo a por defecto de banda
            onSend = false;         //enciende auto
            dialog.dismiss();       //cierro diálogo
        }

    });
    b.setPositiveButton("CANCEL", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            onSend = false;         //enciende auto
            dialog.dismiss();
        }
    });
    b.show();
}
```

Y el aspecto es:

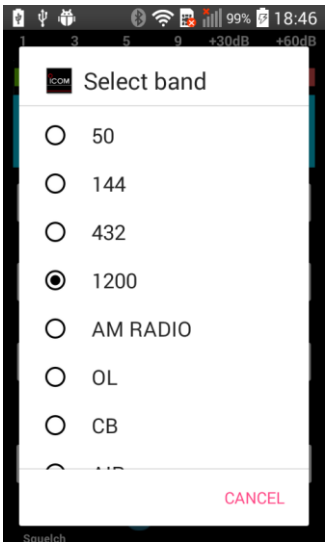


Ilustración 60: Menú seleccionar banda

3.2.m) Extra controls

La segunda opción que hemos preparado en el menú lateral es para incorporar un control de ganancia de micro y la potencia de salida de radiofrecuencia. Aunque en el equipo real son potenciómetros circulares se ha decidido hacerlo en nuestra App por medio de SeekBars, ya que es más cómodo.

El código ha quedado:

```
protected void showDialogExtraC() {
     LayoutInflater layoutInflater = LayoutInflater.from(MainActivity.this);
     View promptView = layoutInflater.inflate(R.layout.extrac_dialog, null);
     AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(
             MainActivity.this);
     alertDialogBuilder.setView(promptView);
     //seekBars micGain y rfGain
     SeekBar mMicSbar, mRfSbar;
     mMicSbar = (SeekBar) promptView.findViewById(R.id.micSeekBar);
     mRfSbar = (SeekBar) promptView.findViewById(R.id.rfSeekBar);
     mMicSbar.setProgress((int) (Double.parseDouble(mMicGain)));
     mRfSbar.setProgress((int) (Double.parseDouble(mRfPower)));
     //micGain
     mMicSbar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
         @Override
         public void onStopTrackingTouch(SeekBar micsBar) {
             esperarAuto(200);
         }
         @Override
         public void onStartTrackingTouch(SeekBar micsBar){
             onSend = true; //desactiva automático
         }
         @Override
         public void onProgressChanged(SeekBar micsBar, int progress, boolean fromUser)
         {
             String micComando = String.valueOf(micsBar.getProgress());
             micComando = "140B" + formatAx(micComando, 4); //Mic => 14 0B [0000-0255]

             try {
                 byte[] mComando = decodeHex(micComando);
                 if (btConnected && !bloqueado) mConnectedThread.write(mComando);
                 turno = 19; //para que auto lea mic/rf primero
             } catch (DecoderException de) {
                 Log.e("TAG", "Exception: " + "de " + "mic gain");
             }
         }
     });
     //rfPower
     mRfSbar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
         @Override
         public void onStopTrackingTouch(SeekBar rfsBar) {
             esperarAuto(200);
         }
         @Override
         public void onStartTrackingTouch(SeekBar rfsBar){
             onSend = true; //desactiva automático
         }
         @Override
         public void onProgressChanged(SeekBar rfsBar, int progress, boolean fromUser) {
             String rfComando = String.valueOf(rfsBar.getProgress());
             rfComando = "140A" + formatAx(rfComando, 4); //RF => 14 0A [0000-0255]

             try {
                 byte[] rComando = decodeHex(rfComando);
                 if (btConnected && !bloqueado) mConnectedThread.write(rComando);
                 turno = 19; //para que auto lea mic/rf primero
             } catch (DecoderException de) {
                 Log.e("TAG", "Exception: " + "de " + "Rf Power");
             }
         }
     });
     alertDialogBuilder.setCancelable(false)
         .setPositiveButton("Exit", new DialogInterface.OnClickListener() {
```

```
            public void onClick(DialogInterface dialog, int id) {
                dialog.cancel();
                mDrawer.closeDrawer();  //cierro menú lateral
            }
        })
        .setNegativeButton("Cancel",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                }
         });
    AlertDialog alert = alertDialogBuilder.create();
    alert.show();
}
```

Y el resultado:

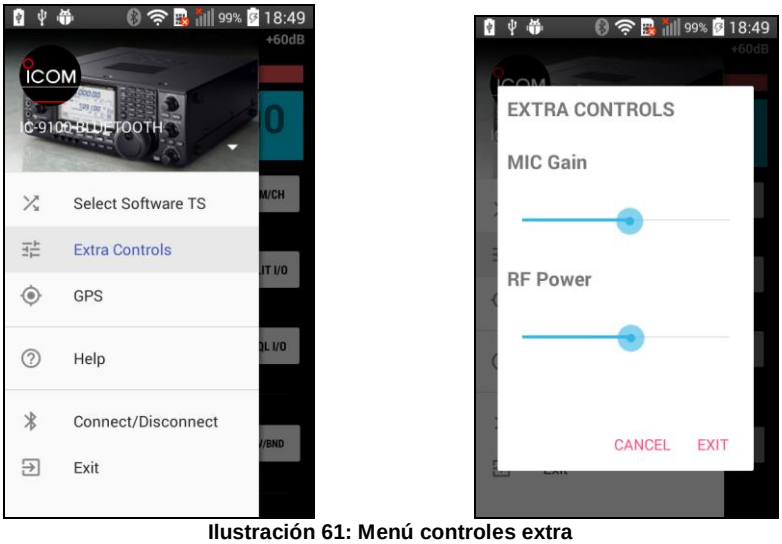


Ilustración 61: Menú controles extra

3.2.n) Posicionaminto de red

La tercera opción del menú lateral es el GPS, aunque realmente internamente vamos a usar la localización de red. El motivo es que la utilidad que vamos a incorporar es la de mandar al equipo de radio nuestras coordenadas, ya que está preparado para ello. En efecto, el Icom IC-9100 está equipado de un conector trasero para conectarle un GPS externo y de esta manera en el modo de modulación digital DV podemos saber entre otras cosas la distancia hasta nuestros interlocutores, y como existe un comando para poder mandarle las coordenadas lo vamos a emplear para poder usar tal funcionalidad desde nuestro dispositivo móvil, ahorrándonos el GPS externo. Normalmente el equipo va a estar en un sitio cerrado ya que es muy grande, por lo que se justifica que sea localización de red y no GPS. El código queda:

```
protected void showDialogGPS() {
    onSend = true; //desactiva automático
    LayoutInflater layoutInflater = LayoutInflater.from(MainActivity.this);
    View promptView = layoutInflater.inflate(R.layout.gps_dialog, null);
    AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(
            MainActivity.this);
    alertDialogBuilder.setView(promptView);

    final TextView textLat, textLong;
    Button mBtnRefresh, mBtn00;
    textLat = (TextView)promptView.findViewById(R.id.textViewLat1);
    textLong = (TextView)promptView.findViewById(R.id.textViewLong1);
```

```
mBtnRefresh= (Button)promptView.findViewById(R.id.buttonRefresh1);
mBtn00= (Button)promptView.findViewById(R.id.button001);

alertDialogBuilder.setCancelable(false)
    .setPositiveButton("Send to RTX", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {

            if(textLat.getText().length()>=3 && textLong.getText().length()>=3) {
                double lat = Double.valueOf(textLat.getText().toString());
                double lon = Double.valueOf(textLong.getText().toString());

                double lat_deg = Math.floor(Math.abs(lat));//necesito parte entera
                double lat_min = (Math.abs(lat) - lat_deg)*60;//nec. p.entera y 3dec

                String sLat_deg = String.valueOf(lat_deg);
                String sLat_min = String.valueOf(lat_min);
                String signo;
                if (lat_deg > 0) signo = "1"; else signo = "0"; //1=N, 0=S
                String grados = sLat_deg.substring(0, sLat_deg.indexOf('.'));
                if (grados.length() == 1) grados = "0" + grados;
                String minutos = sLat_min.substring(0, sLat_min.indexOf('.'));
                if (minutos.length() == 1) minutos = "0" + minutos;
                String decimales = sLat_min.substring(sLat_min.indexOf('.') + 1,
                        sLat_min.length());
                if (decimales.length()>=3) decimales=decimales.substring(0,3);
                    else if (decimales.length()==2){decimales=decimales+"0";}
                    else if (decimales.length()==1){decimales=decimales+"00";}
                Log.e("TAG", signo + " Lat.Deg: : " + grados + " Lat.Min: " +
                        minutos + " Dec: " + decimales);

                double lon_deg = Math.floor(Math.abs(lon)); //necesito parte entera
                double lon_min = (Math.abs(lon) - lon_deg) * 60;//nec. p.ent. y 3dec

                String sLon_deg = String.valueOf(lon_deg);
                String sLon_min = String.valueOf(lon_min);
                String signo2;
                if (lon_deg > 0) signo2 = "1"; else signo2 = "0"; //1=E, 0=W
                String grados2 = sLon_deg.substring(0, sLon_deg.indexOf('.'));

                if (grados2.length() == 2) grados2 = "0" + grados2;
                if (grados2.length() == 1) grados2 = "00" + grados2;
                String minutos2 = sLon_min.substring(0, sLon_min.indexOf('.'));
                if (minutos2.length() == 1) minutos2 = "0" + minutos2;
                String decimales2 = sLon_min.substring(sLon_min.indexOf('.') + 1,
                        sLon_min.length());
                if (decimales2.length()>=3) decimales2=decimales2.substring(0,3);
                    else if (decimales2.length()==2){decimales2=decimales2+"0";}
                    else if (decimales2.length()==1){decimales2=decimales2+"00";}
                Log.e("TAG", signo2 + " Lon.Deg: : " + grados2 + " Lon.Min: " +
                        minutos2 + " Dec: " + decimales2);
                String gpsComando = "";
          gpsComando = "1A050158";
          gpsComando = gpsComando + grados + minutos + decimales + "00" + signo;
          gpsComando = gpsComando + "0"; //delimitador fijo
          gpsComando = gpsComando + grados2 + minutos2 + decimales2 + "00" + signo2;
                try {
                    byte[] coorComando = decodeHex(gpsComando);
                    if (btConnected && !bloqueado)
                        mConnectedThread.write(coorComando);
                } catch (DecoderException de) {
                    Log.e("TAG", "Exception: " + "de " + "gps");
                }
            }
            onSend = false;          //enciende auto
            dialog.cancel();
            mDrawer.closeDrawer();  //cierro menú lateral
        }
    })
    .setNegativeButton("Cancel",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                onSend = false;
                dialog.cancel();
            }
    });
AlertDialog alert = alertDialogBuilder.create();
alert.show();
```

```
    //localización de red
    final LocationManager locationManager =(LocationManager) mContext.
            getSystemService(Context.LOCATION_SERVICE);
    if (locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {
           String[] mLaLon =  calcularLatLon(locationManager);
           textLat.setText(mLaLon[0]);
           textLong.setText(mLaLon[1]);
    }else {
        Log.e("TAG ", "ERROR: Location disabled ");
        Toast.makeText(getApplicationContext(), "Enable location please",
                Toast.LENGTH_SHORT).show();
        alert.cancel();
    }
    mBtnRefresh.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            String[] mLaLon =  calcularLatLon(locationManager);
            textLat.setText(mLaLon[0]);
            textLong.setText(mLaLon[1]);
        }
    });
    mBtn00.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            textLat.setText("0.0");
            textLong.setText("0.0");
        }
    });
}
// método para obtener localización
public String[] calcularLatLon(LocationManager locationManager) {
    String[] mLaLon = new String[2];
    mLaLon[0]=""; mLaLon[1]="";
    if (!bloqueado) {
        if (ActivityCompat.checkSelfPermission(getApplication(),
android.Manifest.permission.
                ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
            Location location = locationManager.getLastKnownLocation(
                    LocationManager.NETWORK_PROVIDER);
            if (location != null) { //si he conseguido localización
                String latitud = String.valueOf(location.getLatitude());
                String longitud = String.valueOf(location.getLongitude());
                Log.e("TAG ", "Latitud: " + latitud + " / Longitud: " + longitud);
                mLaLon[0] = latitud;
                mLaLon[1] = longitud;
            } else {
                Log.e("TAG ", "ERROR: Location is null, try again");
                Toast.makeText(getApplicationContext(), "Location not available yet, " +
                        "try again", Toast.LENGTH_SHORT).show();
            }
        } else {
            Log.e("TAG ", "ERROR: No permisos de localización");
        }
    }
    return (mLaLon);
}
```

El resultado es:

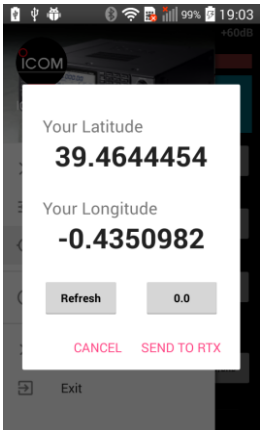


Ilustración 62: Pantalla de localización de red

Vemos ahora como al pulsar SEND TO RTX aparece la localización en la pantalla lcd de la emisora, ya que se ha enviado por comando Bluetooth.

Ilustración 63: Mandando coordenades

El código de conversión de datos de localización ha salido algo extenso, pero es debido a la complejidad del formato del comando. Lo vemos seguidamente:

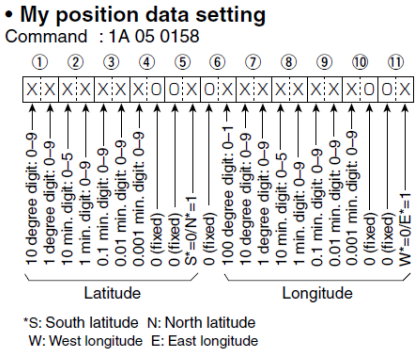


Ilustración 64: Protocolo CI-V 5

3.2.o) Ayuda (créditos)

Ponemos en el menú lateral la opción de buscar ayuda, aunque realmente son créditos con la información de contacto y un enlace URL para más información.

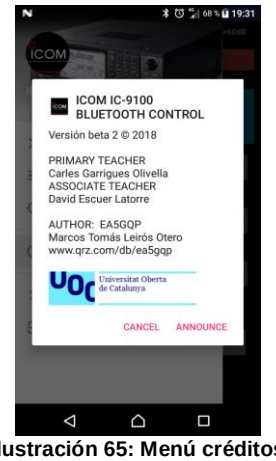


Ilustración 65: Menú créditos

Como podemos observar se ha añadido el botón [ANNOUNCE] en la ayuda.

El transceptor en cuestión posee un sintetizador de voz en Inglés para indicar hablando la frecuencia el modo y la señal (los tres parámetros mas importantes) disponibles en ese momento, es muy usado por personas invidentes. Estudiando los comandos se ha encontrado el 13h 00h que lo pone en marcha, por tanto al pulsar se escucharán tales datos mediante voz real. El código ha quedado:

```
b.setPositiveButton("ANNOUNCE", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        byte[] announceComando = {(byte) 0x13, (byte) 0x00};//announce => 1300
        if (btConnected && !bloqueado) mConnectedThread.write(announceComando);

        onSend = false;          //enciende auto
        dialog.dismiss();
        mDrawer.closeDrawer();  //cierro menú lateral
    }
});
```

3.2.p) Selección de tono T y Tsql (CTCSS)

Se accede a estas dos funciones mediante pulsación larga a [TONE I/O] o [TSQL I/O] respectivamente. Se emplea para superponer un tono inaudible a la transmisión o recepción de manera que el que nos reciba si no tiene programada la misma frecuencia de tono no nos escuchará. Tenemos disponibles los siguientes 50 tonos.

Tabla de subtonos CTCSS: 50 tonos

No	Freq.(Hz)	No	Freq.(Hz)	No	Freq.(Hz)
01	67.0	18	118.8	35	183.5
02	69.3	19	123.0	36	186.2
03	71.9	20	127.3	37	189.9
04	74.4	21	131.8	38	192.8
05	77.0	22	136.5	39	196.6
06	79.7	23	141.3	40	199.5
07	82.5	24	146.2	41	203.5
08	85.4	25	151.4	42	206.5
09	88.5	26	156.7	43	210.7
10	91.5	27	159.8	44	218.1
11	94.8	28	162.2	45	225.7
12	97.4	29	165.5	46	229.1
13	100.0	30	167.9	47	233.6
14	103.5	31	171.3	48	241.8
15	107.2	32	173.8	49	250.3
16	110.9	33	177.3	50	254.1
17	114.8	34	179.9		

Ilustración 66: Tabla de tonos ctcss

Por tanto abrimos un diálogo donde al pulsar en uno de ellos se coge de un vector el valor de la frecuencia del tono y se manda un comando a la emisora para ponerlo. El comando para transmisión es el 1B00xxxx, donde xxxx es el tono cogido del array, y el de recepción es el 1B01xxxx.

El código para gestionar el evento queda así (solo mostramos Tsql):

```
btnTsqlIo.setOnLongClickListener(new View.OnLongClickListener(){
    @Override
    public boolean onLongClick(View v) {
        showSpinnerTtsql("TSQL");
        return true;
    }
});
```

Y la gestión del dialog:

```
protected void showSpinnerTtsql(final String param) {
    AlertDialog.Builder b = new AlertDialog.Builder(this);
    onSend = true;   //apaga auto

    final String[] options = {
            "67.0","69.3","71.9","74.4","77.0","79.7","82.5","85.4","88.5","91.5",

"94.8","97.4","100.0","103.5","107.2","110.9","114.8","118.8","123.0","127.3",

"131.8","136.5","141.3","146.2","151.4","156.7","159.8","162.2","165.5","167.9",

"171.3","173.8","177.3","179.9","183.5","186.2","189.9","192.8","196.6","199.5",

"203.5","206.5","210.7","218.1","225.7","229.1","233.6","241.8","250.3","254.1"};
    final String[] options2 = {

"067.0","069.3","071.9","074.4","077.0","079.7","082.5","085.4","088.5","091.5",

"094.8","097.4","100.0","103.5","107.2","110.9","114.8","118.8","123.0","127.3",

"131.8","136.5","141.3","146.2","151.4","156.7","159.8","162.2","165.5","167.9",

"171.3","173.8","177.3","179.9","183.5","186.2","189.9","192.8","196.6","199.5",

"203.5","206.5","210.7","218.1","225.7","229.1","233.6","241.8","250.3","254.1"};

    int elem=1; //para mostrar el T/TSQL actual
    if (param.equals("T")){
        b.setTitle("Select subtone T\n(CTCSS)");    //muestra T actual
        for(int i=0; i<options2.length; i++){if(options2[i].equals(mT)){elem=i;}}
    }
    else if(param.equals("TSQL")) {
        b.setTitle("Select subtone TSQL\n(CTCSS)"); //muestra TSQL actual
        for(int i=0; i<options2.length; i++){if(options2[i].equals(mTSQL)){elem=i;}}
    }

    b.setIcon(R.mipmap.ic_launcher);
    b.setSingleChoiceItems(options, elem, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            String mTone = options2[which]; //obtiene tono (con el .)

            if (param.equals("T")) {mT=mTone;}            //memorizo T (con .)
            else if(param.equals("TSQL")) {mTSQL=mTone;}  //memorizo TSQL (con .)

            mTone=mTone.substring(0,3)+mTone.substring(4,5); // quito .

            if (param.equals("T")) {mTone="1B00"+mTone;}        //añade del. comando T
            else if(param.equals("TSQL")) {mTone="1B01"+mTone;} //añade del.comando TSQL

            try {
                byte[] newT = decodeHex(mTone); //lo decodifica como Hex
                if (btConnected && !bloqueado) mConnectedThread.write(newT);
            } catch (DecoderException de) {
                Log.e("TAG", "Exception: " + "de " + "T/Tsql");
            }

            esperarAuto(200); //enciende auto temporizado
            dialog.dismiss();         //cierro diálogo
            mDrawer.closeDrawer();    //cierro menú lateral
        }
    });
    b.setPositiveButton("CANCEL", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            onSend = false;           //enciende auto
            dialog.dismiss();
        }
    });
    b.show();
}
```

El resultado es:

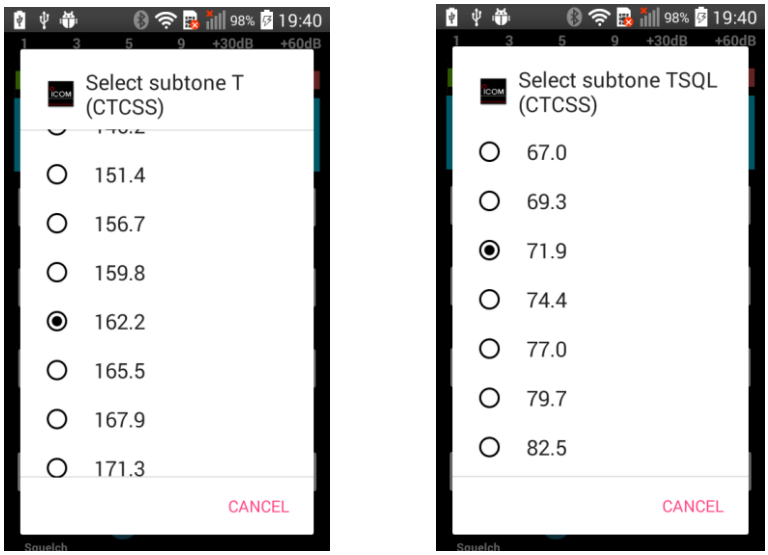


Ilustración 67: Menú de tonos T y Tsql

3.2.q) Duplex Offset

A veces es necesario que la frecuencia de transmisión y recepción no sean la misma, y que al pulsar el PTT para transmitir se cambie automáticamente, es un uso habitual para poder usar repetidores de radio.

Por ejemplo uno de los repetidores de radioaficionado del Monte Bartolo (Desierto de las Palmas – Benicasim – Castellón) transmite en 145.700 MHz, pero recibe en 145.100 MHz por lo que se dice que tiene un duplex offset de -0.600MHz. Otro ejemplo es el de la URE Barcelona en el Tibidabo, que emite en 439.125 MHz pero recibe -7.600 MHz por debajo.

Es evidente pues que poder cambiar ese offset es importante, ponemos ahora la opción de poder verlo y cambiarlo haciendo pulsación larga en [DUP-/SIM].

Comenzamos preparando el duplex_dialog.xml, y el método showDialogDupOf() para cargarlo en la pantalla.
La idea es poder cambiarlo directamente por teclado, pero con la funcionalidad de poder ver el actual en el mismo EditText.

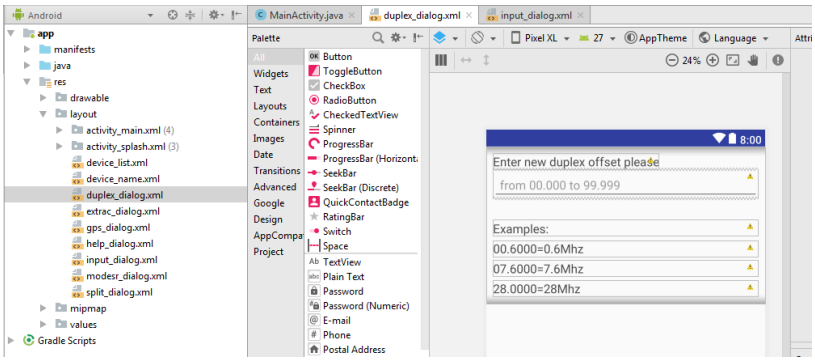


Ilustración 68: Layout dúplex

Lo cargamos para probar y ver el aspecto.

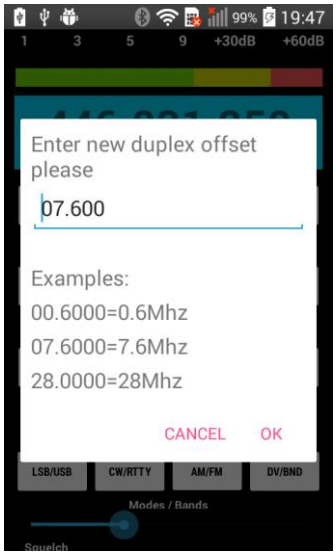


Ilustración 69: Menú dúplex

Para ello hemos añadido el siguiente código:

```
protected void showDialogDupOf() {
    onSend = true;

    // get prompts.xml view
    LayoutInflater layoutInflater = LayoutInflater.from(MainActivity.this);
    View promptView = layoutInflater.inflate(R.layout.duplex_dialog, null);
    AlertDialog.Builder alertDialogBuilder=new AlertDialog.Builder(MainActivity.this);
    alertDialogBuilder.setView(promptView);

    final EditText editText = (EditText) promptView.findViewById(R.id.edittext);
    editText.setText(mDuOf); //muestra duplex offset actual

    // setup a dialog window
    alertDialogBuilder.setCancelable(false)
       .setPositiveButton("OK", new DialogInterface.OnClickListener() {
              public void onClick(DialogInterface dialog, int id) {

                  String cad = editText.getText().toString();
                  if (cad.length() == 7 && cad.substring(2,3).equals(".")){ //xx.xxxx
                      cad=cad.substring(5,6)+ cad.substring(6,7)+cad.substring(3,4)+
                              cad.substring(4,5)+ cad.substring(0,1)+cad.substring(1,2);
                      cad = "1A050017" + cad; // compongo comando + datos

                      try {
                        byte[] nDupOff = decodeHex(cad);
                        if (btConnected && !bloqueado) mConnectedThread.write(nDupOff );
                      } catch (DecoderException de) {
                          Log.e("TAG", "Exception: " + "de " + "DupOf");
                      }
                  }else{
                      editText.setError("format xx.xxxx");
                      onSend = false; //activa automático
                  }
                   esperarAuto(200); //enciende auto temporizado
                   mDrawer.closeDrawer(); //cierro menú lateral
              }
       })
       .setNegativeButton("Cancel",
            new DialogInterface.OnClickListener() {
                  public void onClick(DialogInterface dialog, int id) {
                       onSend = false;
                       dialog.cancel();
                  }
       });
        // create an alert dialog
        AlertDialog alert = alertDialogBuilder.create();
```

```
        alert.show();
    }
```

3.2.r) Split offset

Otra funcionalidad muy similar a la anterior es el Split Offset, menú que aparecerá al hacer pulsación larga en [SPLIT I/O]. El Duplex se suele usar más en las bandas de VHF y UHF, sin embargo en HF es mas usado el Split. Es muy habitual para hacer DX o comunicaciones a larga distancia especialmente si es una actividad importante y hay muchos radioaficionados llamando a la vez.

La forma de abordarlo es idéntica al punto anterior por lo que está casi todo dicho, a excepción de que el formato del comando retornado es distinto (ver tabla de comandos). Además una diferencia fundamental es que en el split hay que indicar si el offset es positivo o negativo, eso lo solucionamos poniendo un + o bien un - delante, y el rango va de 0.000 a 9.999, vemos el formato de la respuesta del comando que hay que gestionar. Por tanto podemos teclear desde -9.999 (negativo) hasta +9.999 (positivo).

Vemos directamente el aspecto:

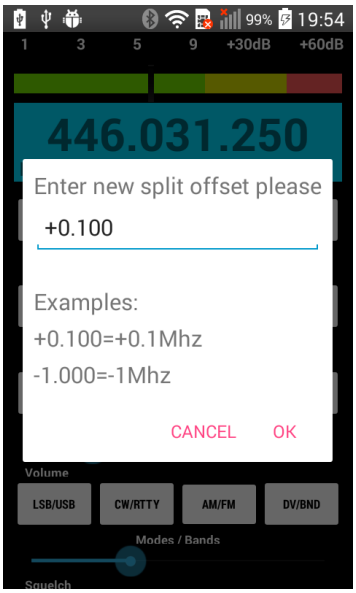


Ilustración 70: Menú split

4. Revisión de la planificación

Con respecto al seguimiento de la planificación se ha seguido con bastante fidelidad, prueba de ello son las entregas antes de las fechas acordadas, no obstante hay algunos cambios que merece la pena comentar.

Para comenzar se han añadido algunas mejoras e incluso funcionalidades que no estaban en la planificación inicial. En cuanto a las funcionalidades nuevas la más importante es la de poder mandar vía comando a la emisora desde el Smartphone la latitud y longitud correspondiente a nuestra localización, así la emisora puede calcular la distancia hasta nuestra interlocutor si este también tiene las coordenadas puestas en la emisora. En cuanto a mejoras podemos destacar el diseño multipantalla.

Por otro lado hay un cambio importante en el diseño propuesto que no podríamos haberlo llevado a cabo en caso de que el proyecto fuera bajo demanda, ya que se ha cambiado el menú planificado del tipo HUB central a un estilo con menú lateral, que una vez probado ha quedado claro que es mas eficiente este segundo ya que evitamos un paso para conectar y conseguir otros resultados de varias acciones.

Analizando todo lo anterior podemos suponer que se han dedicado algunas horas de más con respecto a la planificación, que hemos programado en fin de semana para poder llegar a los plazos de entrega.

5. Conclusiones

Las conclusiones son que solamente mediante la elaboración completa de un proyecto por parte del estudiante, desde la idea inicial hasta la obtención del resultado final, pasando por la planificación, el desarrollo, la depuración y las pruebas es posible integrar todos los conocimientos que componen el Master Oficial en Desarrollo de Aplicaciones para dispositivos Móviles de la UOC.

Reflexionando sobre los resultados creemos que han sido incluso mejor que los esperados, ya que se ha convertido en una verdadera App que puede dar mucho de sí, pues muchos usuarios pueden estar interesados en trabajar con su radio desde otra habitación próxima al conocido como “cuarto de las chispas”, solucionándole por tanto IcomBTC la situación.

Con respecto a posibles líneas de trabajo futuro que no se han podido explorar tenemos varias posibles, en primer lugar podemos ampliar las funcionalidades ya que se nos han quedado decenas de comandos posibles sin usar como por ejemplo el control del atenuador o de los previos de recepción, en segundo lugar podemos añadir un menú de configuración para permitir que la app valga para equipos similares como el Icom IC-7100, y en tercer lugar sería de gran interés cambiar el diseño hardware del interface Bluetooth para que tengamos audio tanto de ida como se vuelta en nuestro Smartphone. En este último sentido se ha encontrado recientemente el módulo Microchip RN52 que posee tanto datos UART como audio, ya hemos encargado uno.

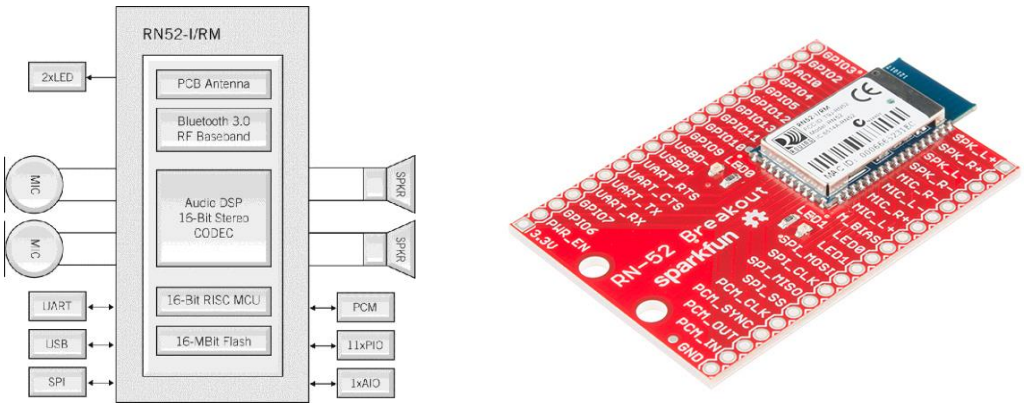


Ilustración 71: Módulo Bluetooth RN52 (Audio+Datos)

6. Glosario

Arduino. Marca de un plataforma de open hardware italiana, basada en una placa con componentes electrónicos expansible y programable.

Banda ciudadana. Porción del espectro de frecuencias destinada a la libre comunicación entre el personal civil, por medio de la radio.

Bluetooth. Especificación industrial para Redes Inalámbricas de Área Personal (WPAN) creado por *Bluetooth* Special Interest Group, Inc.

CI-V. Protocolo de comunicación serie propietario de Icom INC.

Dúplex. Sistema capaz de transmitir en una frecuencia y recibir en otra.

PMR. Personal Mobile Radio, transceptores de 446mhz sin licencia

Radioaficionado. Persona amante de la radio y que está autorizada para emitir y recibir mensajes radiados privados, usando bandas de frecuencia jurídicamente establecidas.

Simplex. Sistema que emite y recibe en la misma frecuencia.

Subtonos CTCSS. Sistema Silenciador Controlado por Tono Continuo.

TS. Tuning Step, frecuencia de salto programada al subir o bajar el dial.

7. Bibliografía

Libro: **Programación con Android**
Edición 2016. Bill Phillips y otros. Anaya Multimedia.
ISBN: 8441537488 ISBN-13: 9788441537484

Web: **Conexión bluetooth Android con Arduino**
http://cursoandroidstudio.blogspot.com.es/2015/10/conexion-bluetooth-android-con-arduino.html

Web: **Custom Rotary Knob Control for Android**
https://www.pocketmagic.net/custom-rotary-knob-control-for-android/

Web: **Custom seek bar with customizable multiple color on background**
https://azzits.wordpress.com/2013/11/17/customseekbar/

Web: **Material Drawer By Mike Penz**
https://github.com/mikepenz/MaterialDrawer

Web: **Equivalent to a JavaScript setInterval/setTimeout in Android/Java**
https://stackoverflow.com/questions/4817933/what-is-the-equivalent-to-a-javascript-setinterval-settimeout-in-android-java

Web: **Detect double button click in Android Studio**
https://www.youtube.com/watch?v=PEpQT4x-kKc

Web: **HC-06 Bluetooth Module Slave With Adapter**
http://cursoarduinomega.blogspot.com/2015/09/hc-06-bluetooth-module-slave-with.html

8. Anexos (en documentos aparte)

Manual de instrucciones del Icom IC-9100
IC-9100_Instruction_Manual.pdf

Manual de comandos Icom CI-V
ICOM_CI-V_manual.pdf

Esquema y PCB del Interface hardware
EsquemaYplaca.pdf

Manual de la App Icom BTC
ICOM_IC-9100_BLUETOOTH_CONTROL_Manual.pdf