UNIVERSIDAD DE LEÓN
CURSO ACADÉMICO 2009 / 2010

Trabajo Fin de Carrera

SIMULACIÓN DE SISTEMA DE BÚSQUEDA DE NÁUFRAGOS DF935 (DIRECTION FINDER)

DOCUMENTACIÓN

Jose Alberto Benítez Andrades
Trabajo de Fin de Carrera
Escuela de Ingenierías Industrial e Informática

UNIVERSIDAD DE LEÓN

El presente documento es presentado como Trabajo de Fin de Carrera por **D. Jose Alberto Benítez Andrades** , alumno de la Escuela de ingenierías industrial e Informática de la Universidad de León, con el fin de obtener el título de **INGENIERO EN INFORMÁTICA**. El proyecto **SIMULACIÓN DE SISTEMAS DE BÚSQUEDA DE NÁUFRAGOS DF935 (DIRECTION FINDER)** ha sido tutelado por **D. Luis Panizo Alonso.**

Vº Bº Tutor

Dr. D. Luis Panizo Alonso

Vº Bº Oficina Técnica

Dr. D. Manuel Castejón Limas

Autor

D. Jose Alberto Benítez Andrades

León, 28 de Mayo de 2010

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

0.INTRODUCCIÓN

- Lenguaje: GNU C++
- Dirigido por: Jorge Balbás Rodrigo
- Presencia necesaria en el CES, durante el desarrollo y pruebas
- Detalle:
    - Estudio de la metodología de desarrollo de módulos de simulación de Indra. Programación orientada a objeto bajo entorno Linux.
    - Estudio del equipo real a simular DF935.
    - Desarrollo de interface HMI gráfico simulando el interface real e integración en una consola simulada.
    - Simulación del equipo de detección de náufragos (DF935).
    - Integración de dicho sistema en el simulador de Helicóptero EC225 en el desarrollo con Indra para EC UK.

Figura 0.1: *Dispositivo Chelton DF935.*

El **Grupo Eurocopter** es una compañía fabricante de helicópteros formada en 1992 a partir de la unión de las divisiones de Aérospatiale, Francia, y DaimlerChrysler Aerospace AG, Alemania.

El Grupo Eurocopter es una compañía franco-germana-española totalmente subsidiaría de EADS, sus factorías se encuentran en Francia (Marignagne y La Courneuve), en Alemania (Kassel, Donauwörth y Ottobrunn) y en España (Madrid-Cuatro Vientos y Albacete).[1] Además, Eurocopter se encuentra presente en los cinco continentes por medio de unas 20 empresas subsidiarias o afiliadas distribuidas por todo el mundo, una de ellas es la brasileña Helibrás.

En 2008, Eurocopter ha confirmado su primera posición mundial en la fabricación de helicópteros para los mercados civil y de servicios públicos con una cuota de mercado de 53%. Los productos del Grupo representan actualmente el 30% de la flota mundial de helicópteros. El Grupo emplea a aproximadamente 15.600 personas.

Figura 0.2: *Captura del helicóptero EC225*

Figura 0.3: *Cabina del helicóptero EC225*

1.ESTRUCTURA DEL SIMULADOR DEL HELICÓPTERO EC225

1.1. PI – HOST – CABINA VIRTUAL – CREAL

El simulador de un helicóptero real se compone de las siguientes partes:

- PI (Posición de instructor): Aplicación mediante la cual el instructor selecciona una serie de parámetros de arranque, posición del helicóptero, estado del clima, y otros detalles para comenzar la simulación de un vuelo. Para que funcione correctamente, las personas que realizan la PI, deben conocer la distinta relación de variables de los distintos módulos de la cabina.
- Audio y comunicaciones: Otro departamento se encarga de todo lo relacionado con los sonidos que debe realizar el simulador, ya que, el simulador del helicóptero, simula también los sonidos y las distintas vibraciones del mismo, siendo así, más realista.
- Sistema Visual: Otro departamento se encarga de realizar el sistema visual del simulador, es decir, el terreno donde se va a mover el helicóptero, que recogiendo los datos de latitud, longitud y altitud, van a mostrar en los diferentes paneles, el terreno por el que se mueve el helicóptero.
- Cabina Virtual: Cabina mediante la cual, desde cualquier ordenador, conectado al host, se pueden realizar las diferentes pruebas del os módulos desarrollados, sin necesidad de tener que entrar en la cabina real. Facilita el desarrollo de los módulos, ya que, permite hacer todas las pruebas con gran facilidad y rapidez.
- Cabina Real: Aparato físico real del simulador, donde irán alojados todos los aparatos simulados con todos sus botones, pantallas tft, etc.
- Host: En él se encuentran alojados los distintos módulos del sistema. Todas las partes anteriormente definidas, se conectan al host, que es el que contiene toda la información y todo el software necesario para hacer funcionar el helicóptero y realizar las diferentes misiones.

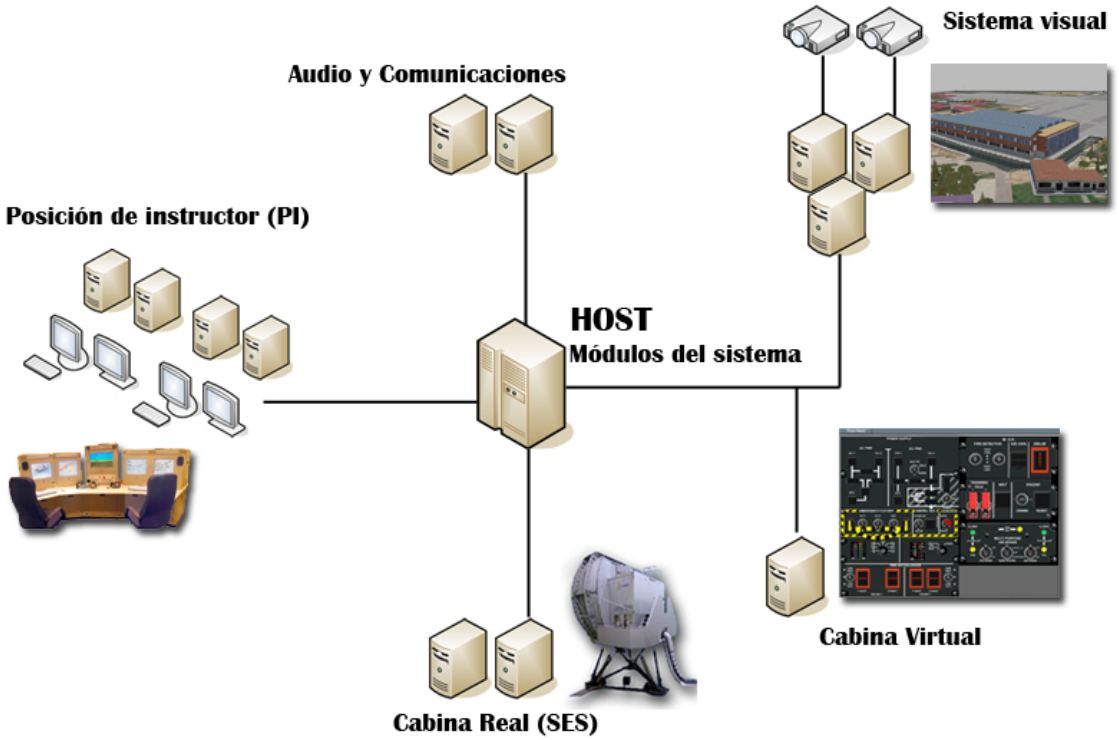


Figura 1.1: *Esquema general del simulador EC225*

1.2. ESQUEMA DE DESARROLLO DE MÓDULOS

Para desarrollar un módulo, se debe seguir un protocolo:

- Desarrollo de módulos: En primer lugar, se realiza la creación de cada uno de los módulos en código c++, definiendo las distintas estructuras y métodos en los ficheros .h y .cpp correspondientes. Paralelamente a este paso, se deben crear los paneles con los que se van a probar los distintos módulos en la cabina virtual.
- Diseño de paneles: Los paneles deben ser diseñados en glStudio, para poder integrar los módulos en la cabina virtual y así hacer las distintas pruebas más adelante.
- Realizados varios tests previos a los distintos módulos, el siguiente paso es integrarlo en la cabina virtual. Una vez integrado en la cabina virtual y testeados por completo, el siguiente paso es realizar las pruebas pertinentes en la cabina real.

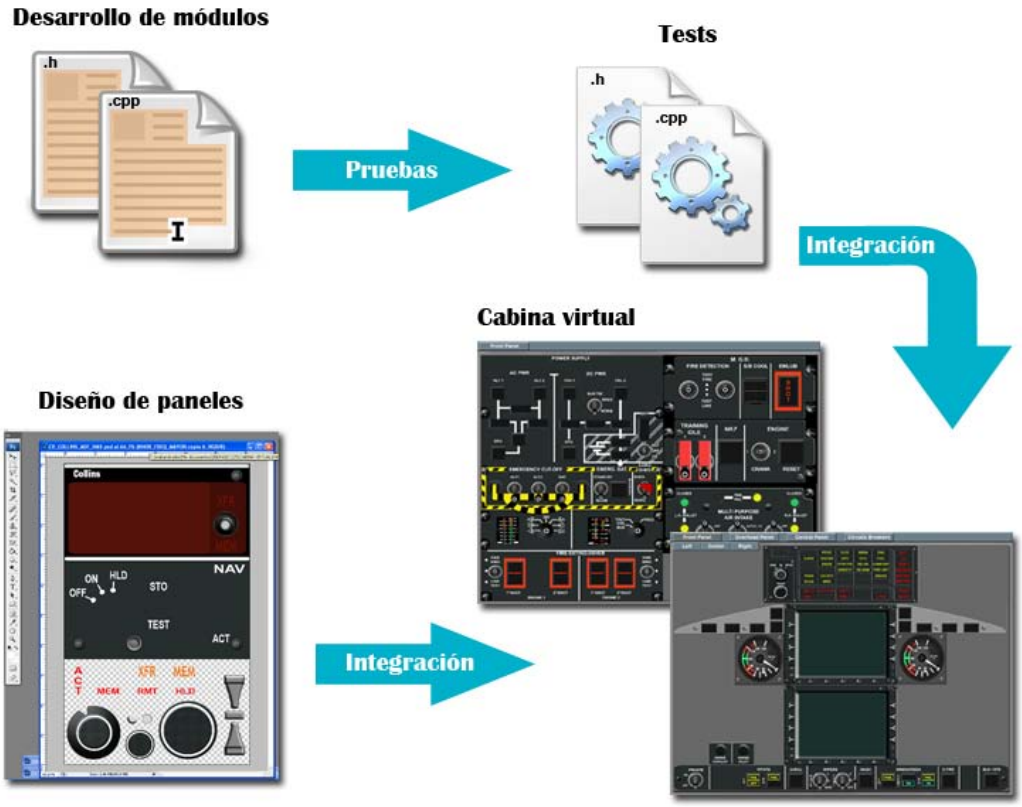


Figura 1.2: *Esquema del desarrollo de módulos*

2. ANÁLISIS PARA EL DESARROLLO DEL SIMULADOR

2.1. ESTUDIO DE LA DOCUMENTACIÓN

Una vez recibida la documentación proporcionada por Indra, se realizó un estudio de los documentos que proporcionaban la información más necesaria para la comprensión del funcionamiento del CHELTON DF 935, aparato que se desea simular.

Con la ayuda del tutor asignado para este proyecto, Miguel Ángel Benítez Andrades, obtuvimos la siguiente información sobre el aparato a simular:

El fichero **Operation Manual** explica de una manera bastante completa el funcionamiento del dispositivo *Chelton DF 935*, describiendo, de manera detallada, los menús por los cuales vamos a poder navegar, funciones que realiza de lo general a lo particular.

2.1.1. DESCRIPCIÓN DEL CHELTON DF 935

El Chelton DF 935, se compone de dos dispositivos que actúan en conjunto, el *Direction Finder* y el *Chelton*.

Es un dispositivo de detección de náufragos y barcos mediante radiobalizas. Dispone de 19 pantallas de menú para la realización de distintas funciones sobre el mismo.

Contiene 6 receptores, 4 de ellos dedicados a la detección de radiobalizas y los 2 restantes, reciben mensajes de tipo DSC y SARSAT con un rango de frecuencias determinadas, proporcionando información sobre cada baliza detectada:

- Edad de la baliza (tiempo en el cual la baliza es detectada).
- Fuerza de la señal recibida (en dB).
- Receptor seleccionado.
- Frecuencia del receptor seleccionado.
- Posición relativa de la baliza respecto al helicóptero.

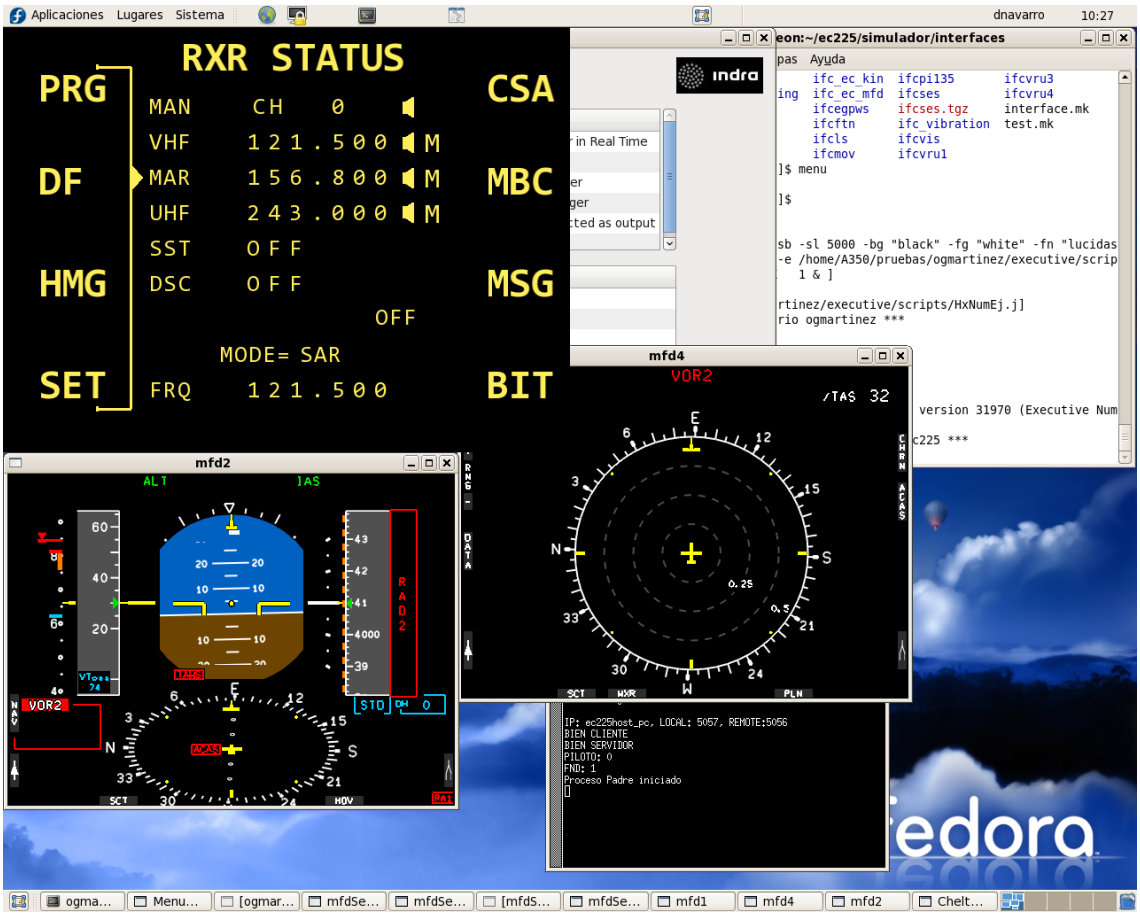


Figura 2.1: *Funcinamiento del chelton y los mfds.*

2.1.2. PANTALLAS DEL CHELTON DF 935

Pantalla 1 – Initial Screen

La primera pantalla inicial, es estática, en ella se muestra información sobre el aparato, como la versión del software *DSP* y del *DF*. Este dispositivo, por defecto, se enciende al iniciarse el helicóptero, a excepción de que no le llegue alimentación, en tal caso, estará apagado. A este dispositivo, le llega una potencia de 28 V de corriente alterna.

Mediante el botón pbKeyPowerButton, que se muestra en la Figura 02, podremos apagar o encender el aparato, presionándolo.

La pantalla inicial, indica la unidad de inicialización del sistema. Esta pantalla es transitoria y no debe ser mostrada el tiempo suficiente para que pueda ser leida, depende de la velocidad de inicialización del *DF*.

Después de un periodo de menos de 5 segundos se observa que el *display* cambia a la pantalla número dos *RXR STATUS* (Receiver Status). Si no es mostrada, la comunicación ha fallado.

Para evitar las posibilidades de que las señales de un desastre natural real sean perdidas, el sistema siempre comienza con los Receptores en su modo *Main* (principal). Cada vez que apagamos y encendemos el dispositivo de nuevo, se cargará la última configuración definida por el usuario.

Pantalla 2 – RXR STATUS

Esta pantalla es utilizada para la selección de receptores; el Receptor Manual *("MAN")* posee diez canales pre-sintonizados; el resto de receptores tienen la posibilidad de cambiar entre *Main ("M)* y *Auxiliar ("A")*; y además se puede ver en ella, el estado de las señales presentes de cada uno de los 6 receptores. Si un receptor está recibiendo una señal, se muestra mostrando el nombre del receptor en modo *inverso*. Mientras esta pantalla es mostrada, el *935-11 DF* continua recogiendo la información de las balizas seleccionadas en los 6 receptores, en las frecuencias mostradas.

Esta pantalla también muestra la información de audio de los distintos receptores (‘’ normal y ‘*en blanco’* en silencio), y cada uno de los mensajes recibidos en el receptor ***COSPAS / SARSAT*** y en el receptor ***DSC*** (*‘B’* para mensajes no leídos y ‘’ para mensajes leídos).

Si un receptor es apagado, entonces se mostrará *'OFF'* en el campo de la frecuencia y si el sistema PTT es activo entonces mostrará *'PTT'* para todos los receptores en el campo de frecuencia.

Las dos últimas líneas de la pantalla muestran en el centro de la pantalla, el modo en el que se está trabajando *SAR* y la frecuencia del receptor *Manual* (notar que esta frecuencia puede ser mostrada también en la línea del receptor *'MAN'*, pero a menudo es conveniente que se muestre el número del canal y la frecuencia simultáneamente).

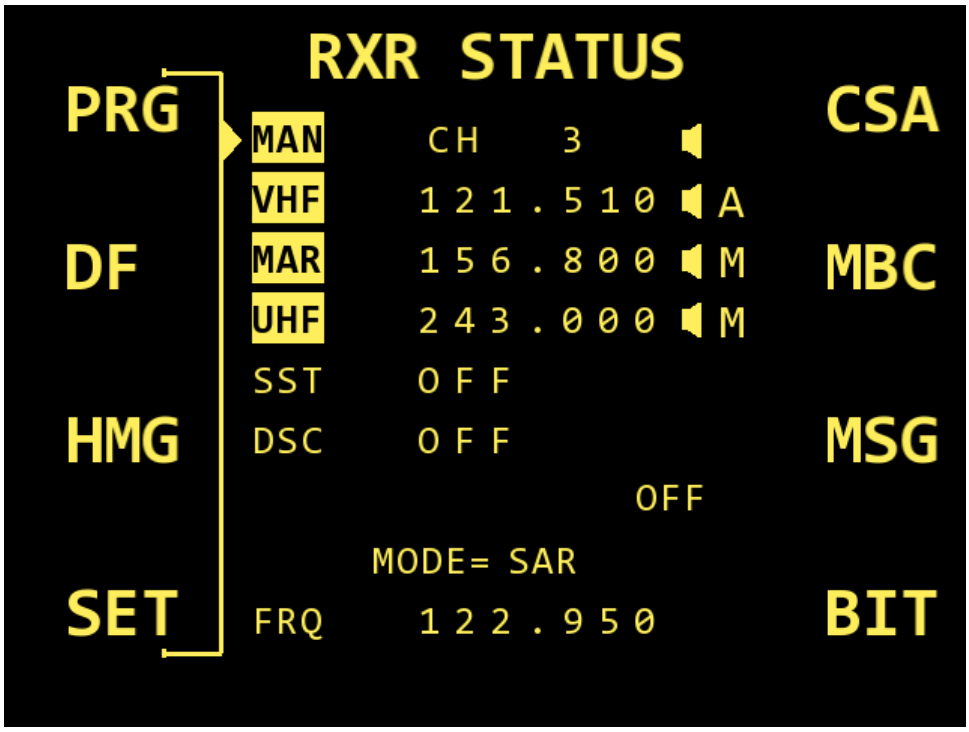


Figura 2.2: *Imagen de la pantalla 'RXR Screen'*

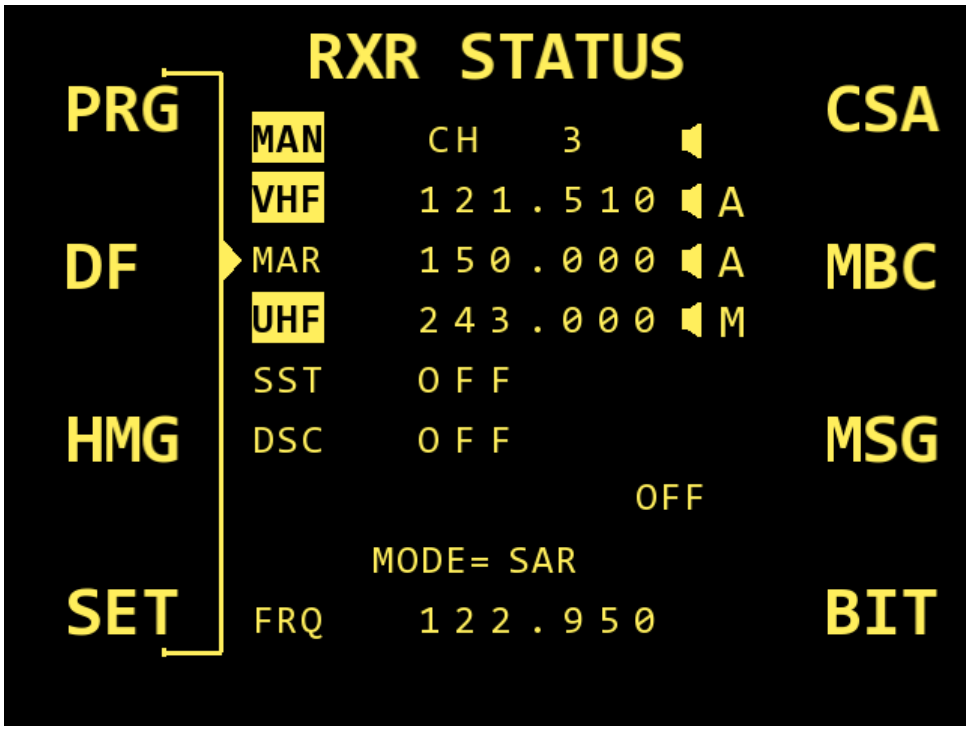
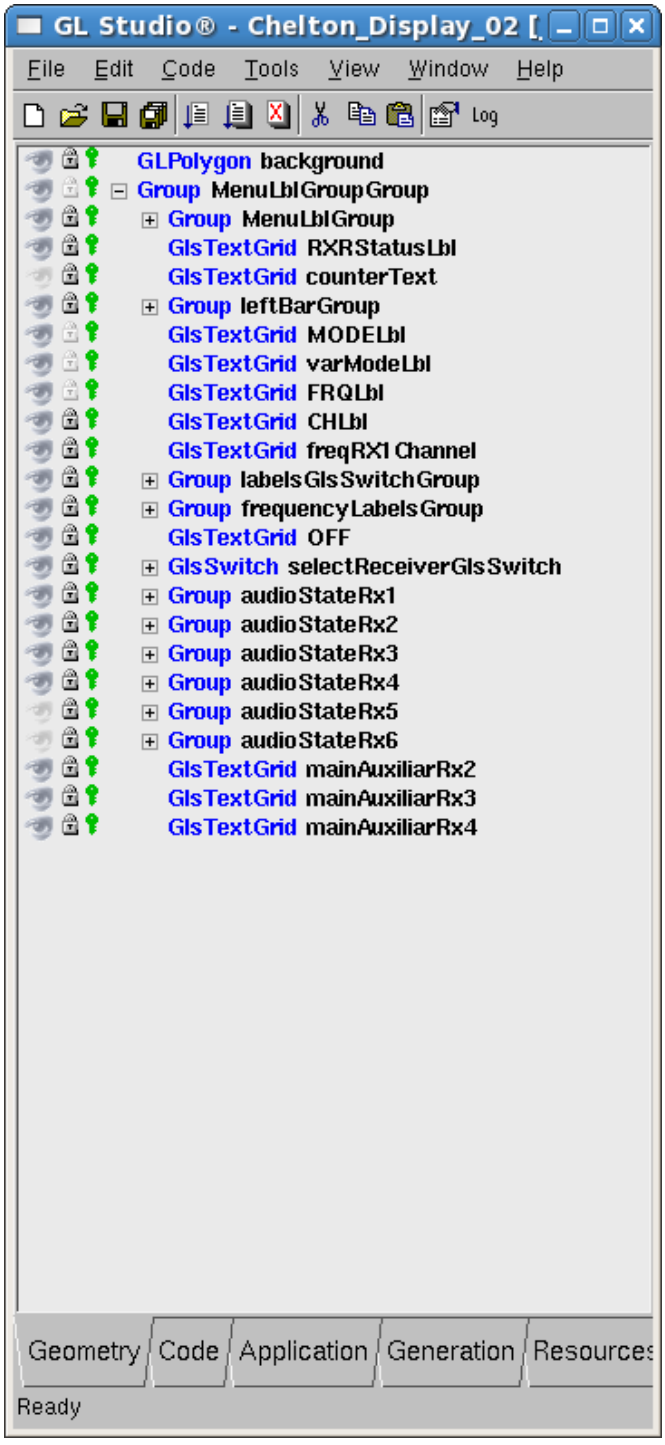


Figura 2.3: *Imagen de la pantalla 'RXR Screen'*

Geometría utilizada



Tras la realización de la imagen completa de la pantalla 2, los elementos creados son:

- Grupo *MenuLblGroup:* Contiene todas las etiquetas del menú de la izquierda y la derecha.
- Grupo *leftBarGroup*: Contiene todo lo relacionado con la barra de la izquierda.
- Grpos *audioStateRx1 a Rx6*: Contienen la parte del audio de cada receptor, el dibujo del altavoz.
- Grupo *frequencyLabelsGroup:* Contiene los textgrids de las distintas frecuencias.
- Grupo *labelsGlsSwitchGroup:* Contiene todos los ítems para poder movernos con el cursor.

Figura 2.4: *Geometría de la pantalla 'RXS Screen'*

Variables utilizadas

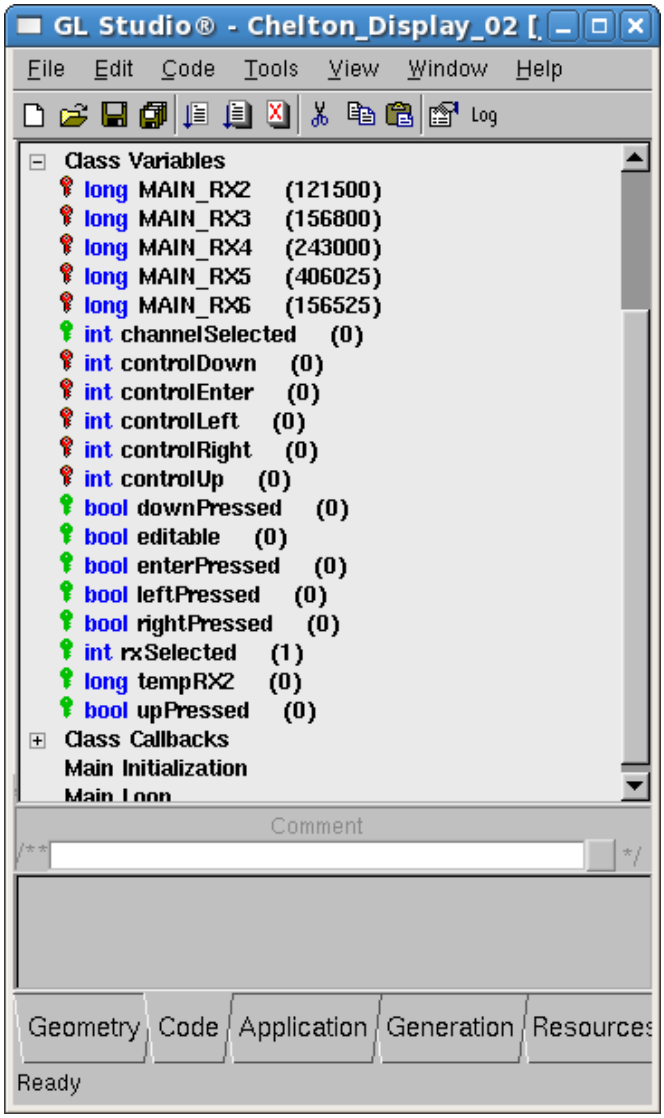


Figura 2.5: *Variables de la pantalla 'RXS Screen'*

Lista de variables utilizadas en la pantalla RXS:

- *MAIN_RX2, MAIN_RX3, MAIN_RX4, MAIN_RX5, MAIN_RX6:* Contienen las frecuencias principales de cada uno de los receptores.
- *channelSelected:* Almacena el canal del receptor uno que esté seleccionado en ese momento.
- *controlDown, controlEnter, controlLeft, controlRight, controlUp:* Variables que controlan que los distintos botones, se pulsen una única vez.
- *downPressed, rightPressed, enterPressed, leftPressed, upPressed:* Variables que almacenan si se pulsa uno de los botones.
- *rxSelected:* Almacena el número de receptor seleccionado.
- *Editable:* Almacena si estamos en modo edición o no.

Métodos utilizados

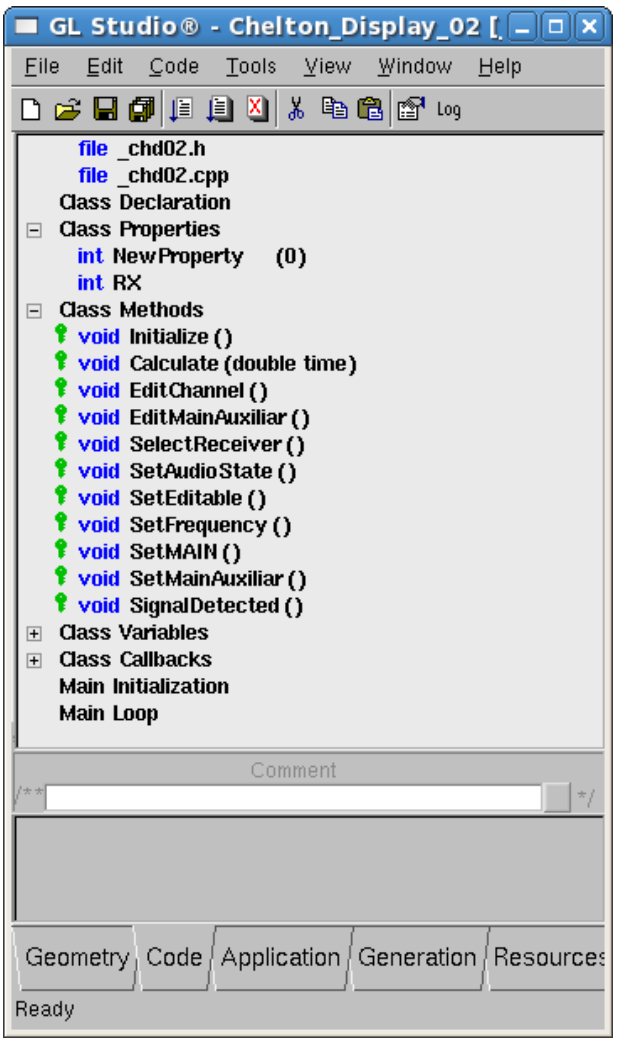


Figura 2.6: *Métodos de la pantalla 'RXS Screen'*

Métodos de la pantalla RXS:

- *EditChannel:* Método encargado de editar el campo del canal en el caso del receptor 1.
- *EditMainAuxiliar:* Cambia entre Principal y auxiliar en cada uno de los receptores.
- *SelectReceiver:* Permite movernos con el cursor por el menú.
- *SetAudioState:* Cambia el estado del audio, entre off y on.
- *SetEditable:* Permite que se pueda editar o no.
- *SetFrequency:* Asigna la frecuencia leyéndola de la variable correspondiente.
- *SetMain:* Ajusta el canal principal.
- *SetMainAuxiliar:* Cambia entre principal y auxiliar.
- *SignalDetected:* Si se detecta una baliza en cualquiera de los receptores, los muestra en inverso.

Pantalla 3 – DF SCREEN

Esta pantalla es usada para mostrar el rumbo (relativo respecto a la plataforma aérea, en nuestro caso concreto, respecto al ***EC225***) del receptor seleccionado, mostrado en la parte superior de la pantalla.

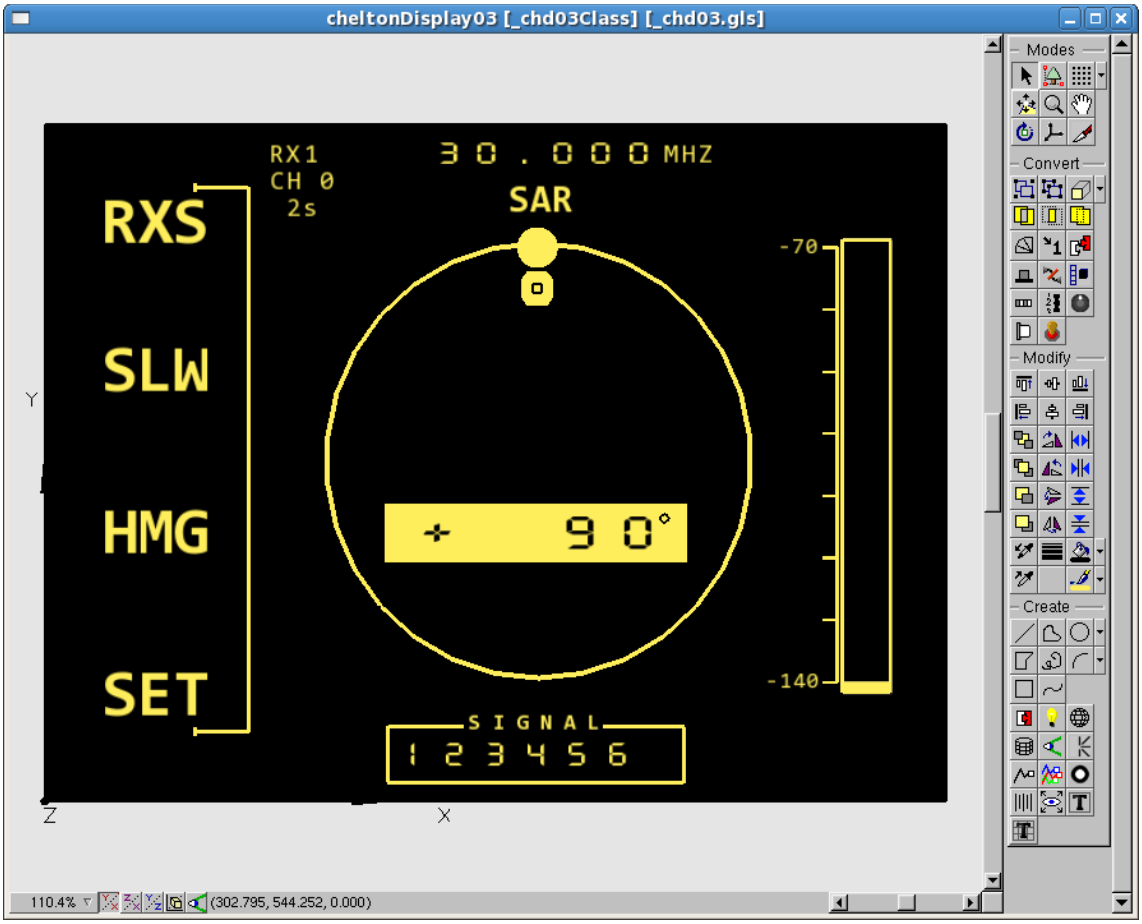


Figura 2.7: *Imagen de la pantalla 'DF Screen'*

Los receptores son numerados del 1 al 6 y tienen las siguientes características:

a. Receptor 1: Este receptor es el Manual, que es totalmente ajustable en un rango de frecuencias desde *30* hasta *470 MHz*. Por conveniencia, el controlador almacena 10 canales pre-programados para este receptor, por lo tanto esta pantalla adicionalmente mostraba el canal que estaba activo y su correspondiente frecuencia asociada.

b. Receptor 2: Este es el receptor para la búsqueda y rescate en el modo *VHF*, que posee como canal principal la frecuencia *121.5 MHz*. Este receptor posee un canal auxiliar programable entre *120.000 MHz* y *130.000 MHz.* Cuando el receptor es seleccionado, la pantalla muestra el número del receptor y su frecuencia asociada.
c. Receptor 3: Este es el receptor para la búsqueda de tipo *Maritime (MAR)* con la frecuencia principal asociada *156.8 MHz*. Este receptor tiene un canal auxiliar programable entre *150.000 MHz* y *160.000 MHz.* Muestra los mismos datos que el receptor 2.
d. Receptor 4: Estes es el receptor para la búsqueda *UHF* con la frecuencia principal asociada de *243 MHz*. Este receptor tiene un canal auxiliar programable entre *240.000 MHz* y *250.000 MHz*. Muestra los mismos datos que el receptor 2.
e. Receptor 5: Este es el receptor para la búsqueda y rescate de *COSPAS / SARSAT* en la frecuencia *406.025 MHz.* Este receptor carece de canal auxiliar.
f. Receptor 6: Este receptor está preparado para la recepción de datos de tipo *Maritime* en el canal 70 con la frecuencia *156.525 MHz*. Este receptor carece de canal auxiliar.

En las líneas inferiores se muestran las señales recibidas por los 6 receptores (un receptor está puesto en modo *inverso* cuando recibe algún tipo de señal).

La fuerza de la señal (en *dBm*) de cada receptor seleccionado en ese momento, es visualizada mediante una barra en la zona lateral derecha de la pantalla.

Geometría utilizada

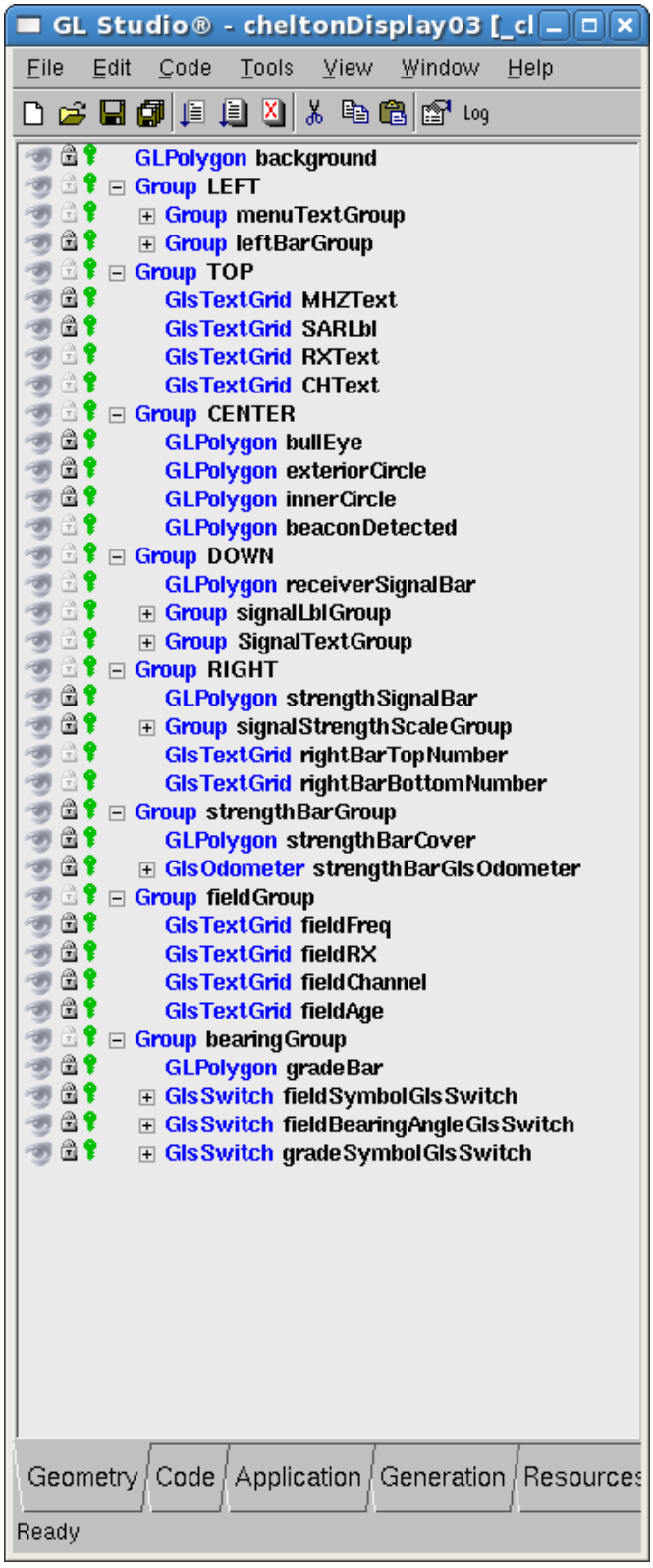


Figura 2.8: *Geometría de la pantalla 'DF Screen'*

Tras la realización de la imagen completa de la pantalla 3, los elementos creados son:

- Grupo *TOP*: **MHz,SAR, RX y CH**, junto con sus campos variables que representan el valor de los mismos, *fieldGroup*.

- Grupo *CENTER*: agrupa todos los elementos que representan las circunferencias en los que se van a mostrar los datos de la baliza detectada.

- Grupo *LEFT*: agrupa los textos del menú que se encuentran a la izquierda y la barra que los agrupa.

- Grupo *DOWN*: agrupa todos los elementos que representan los receptores.

- Grupo *RIGHT:* agrupa los polígonos que representan todo lo relacionado con la barra de fuerza de señal.

- Grupo *fieldGroup*: agrupa todos los campos variables, relacionados con los receptores, frecuencias o canales seleccionados, además del campo edad.

- Grupo *bearingGroup*: agrupa todos los elementos relacionados con el ángulo del rumbo del helicóptero con respecto a la baliza.

Variables utilizadas

Figura 2.9: *Variables de la pantalla 'DF Screen'*

Lista de variables utilizadas en la pantalla DF:

- *SlewFunction*: representa si está, o no, activo, el modo Slew.

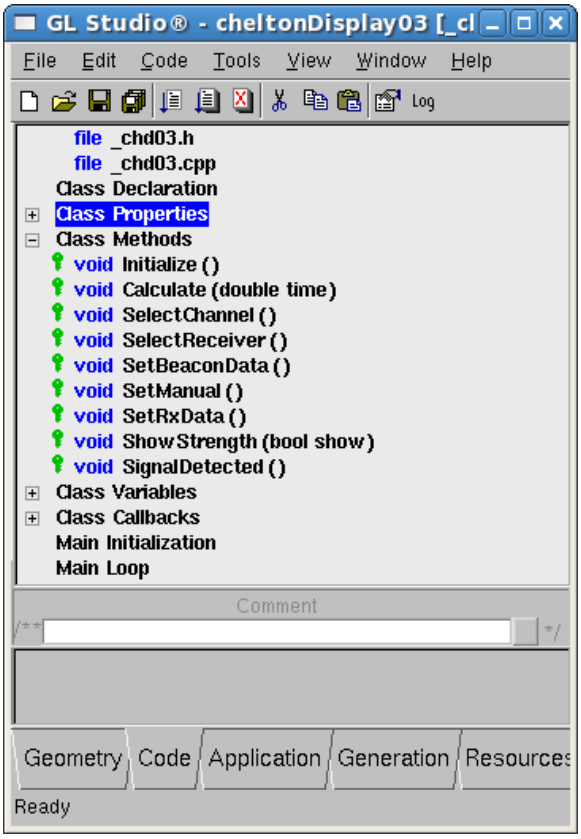
- *channelSelected:* almacena el número de receptor que ha seleccionado el usuario mediante los cursores.

- *controlDown, controlUp, controlRight, controlLeft, controlKey3:* son variables de control de botón (botones de arriba, abajo, izquierda, derecha y 3), con ellas, se consigue controlar que el usuario pulsa un botón, una vez, a pesar de que lo mantenga pulsado.

- *downPressed, upPressed, rightPressed, leftPressed, key3Pressed:* son variables que indican si están o no pulsados los 5 botones correspondientes.

- *iBeacon:* almacena el número de baliza que se ha detectado por el receptor seleccionado.
- *rxSelected*: almacena el número de receptor seleccionado por el usuario (entre 1 y 4).

Métodos utilizados



Lista de métodos creados para el funcionamiento de la pantalla DF:

- *SelectChannel*: método mediante el cual, a través de de la detección de pulsaciones de los botones *Izquierda* y *Derecha*, si nos encontramos en el receptor 1 (Manual), variará entre los canales 0 y 9.

- *SelectReceiver*: método mediante el cual, a través de de la detección de pulsaciones de los botones *Arriba* y *Abajo*, cambiaremos de receptor seleccionado entre 1 y 4, para mostrar la información de las balizas que detecta cada uno.

Figura 2.10: *Métodos de la pantalla 'DF Screen'*

- *SetBeaconData:* método que muestra por pantalla la información de las balizas que son detectadas por un receptor.

- *SetManual:* método que muestra por pantalla el ángulo de rumbo del helicóptero, cuando es editado manualmente por el piloto, activando al función *Slew.*

- *SetRxData*: método que selecciona, si en la pantalla, en el campo *fieldCH*, se debe mostrar el canal seleccionado ('0' … '9') en el caso de que el receptor seleccionado sea el 1, o dos guiones en cualquier otro caso ('- -').

- *ShowStrength:* método que realiza los cálculos necesarios para mostrar la barra de fuerza de señal de la baliza detectada, a la altura que debe estar (apoyado en los datos de latitud y longitud del helicóptero y la baliza, respectivamente).

- *SignalDetected:* método encargado de seleccionar en los distintos switches de señal, de la parte inferior de la pantalla, los receptores que detectan algún tipo de señal. En caso de ser detectada, se mostrará el receptor en modo inverso (número en negro, sobre fondo amarillo).

Pantalla 4 – HOMING SCREEN (HMG)

Esta pantalla es idéntica a la pantalla 2 (*DF Screen*), pero muestra la información de la dirección *izquierda-derecha* con respecto a la dirección hacia la baliza. El rumbo relativo es mostrado también de forma numérica.

En este modo, la pantalla se organiza de tal manera que muestra la información relativa de la baliza de aproximadamente 45” (-45®) hacia la derecha o izquierda.

La línea inferior muestra , al igual que en la pantalla 3, el estado de las señales en los 6 receptores. Cuando uno de ellos detecta una baliza, se puede observar en modo *inverso*.

La señal en *dBm*, también es mostrada mediante una gráfica en la zona derecha de la pantalla.

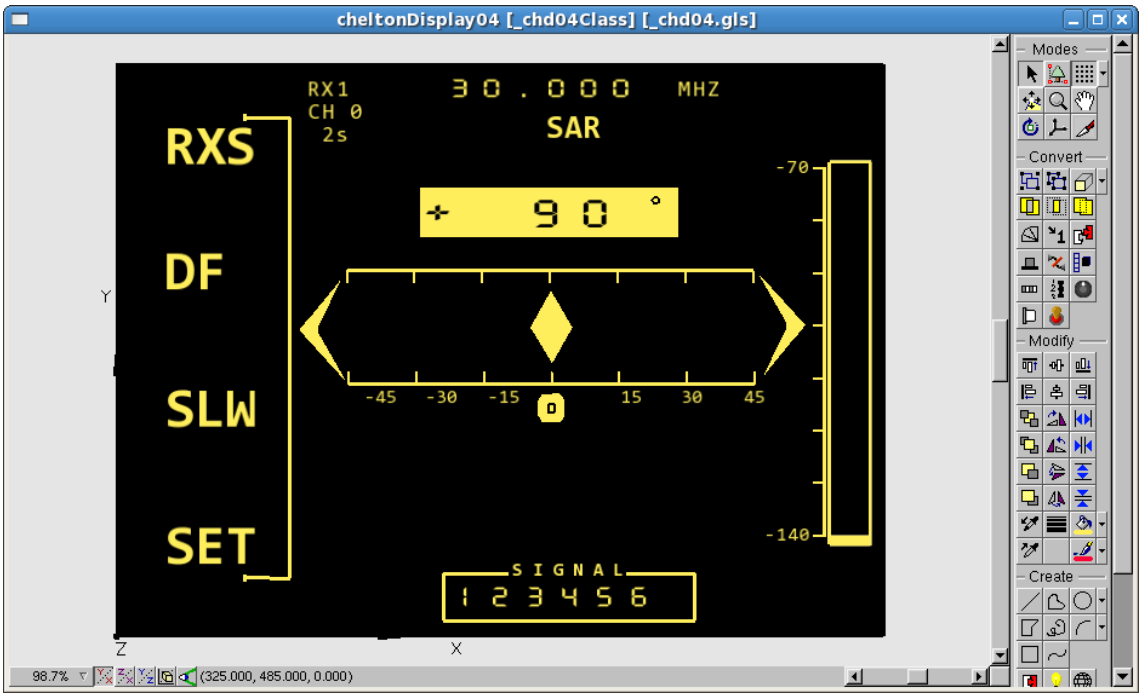


Figura 2.11: *Imagen de la pantalla ‘HMG Screen’*

Geometría utilizada

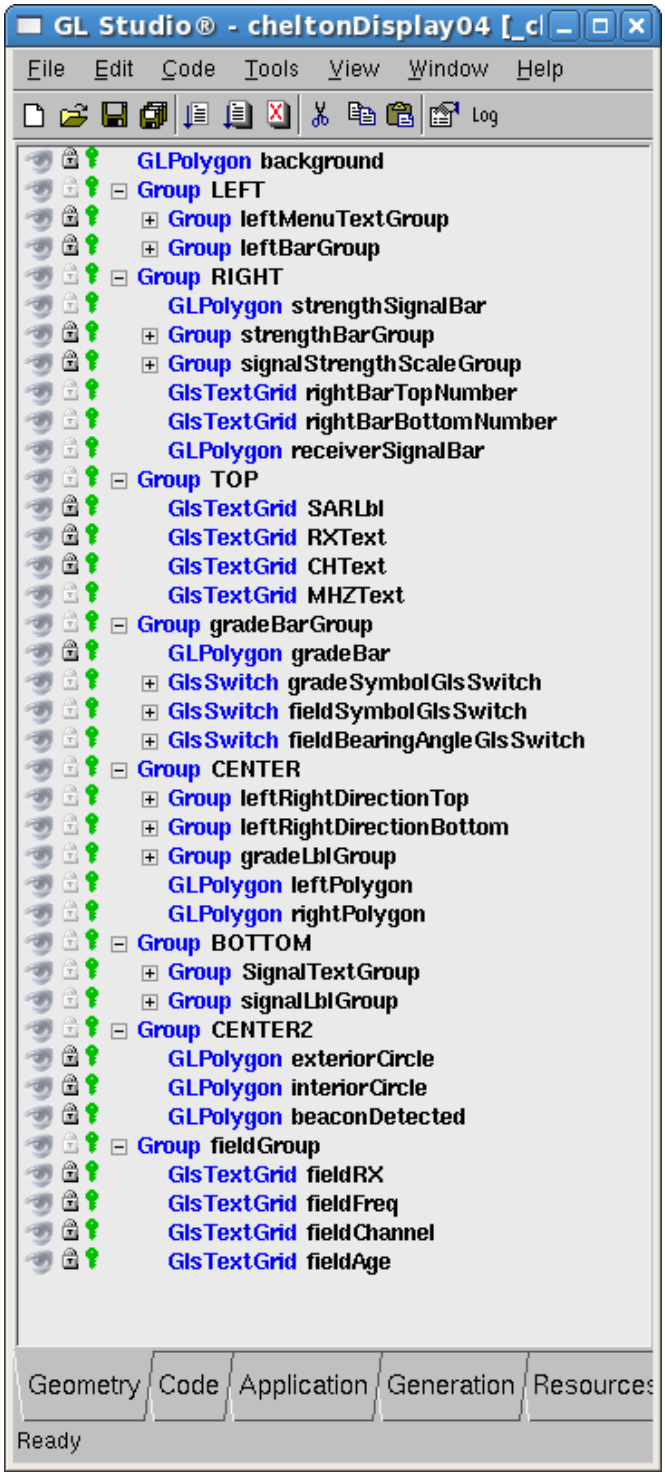


Figura 2.12: *Geometría de la pantalla 'HMG Screen'*

Tras la realización de la imagen completa de la pantalla 3, los elementos creados son:

- Grupo *TOP*: **MHz,SAR, RX y CH**, junto con sus campos variables que representan el valor de los mismos, *fieldGroup*.

- Grupo *CENTER*: agrupa todos los elementos que representan las gráficas del ángulo del rumbo de la baliza.

- Grupo *LEFT*: agrupa los textos del menú que se encuentran a la izquierda y la barra que los agrupa.

- Grupo *DOWN*: agrupa todos los elementos que representan los receptores.

- Grupo *RIGHT:* agrupa los polígonos que representan todo lo relacionado con la barra de fuerza de señal.

- Grupo *fieldGroup*: agrupa todos los campos variables, relacionados con los receptores, frecuencias o canales seleccionados, además del campo edad.

- Grupo *bearingGroup*: agrupa todos los elementos relacionados con el ángulo del rumbo del helicóptero con respecto a la baliza.

Variables utilizadas

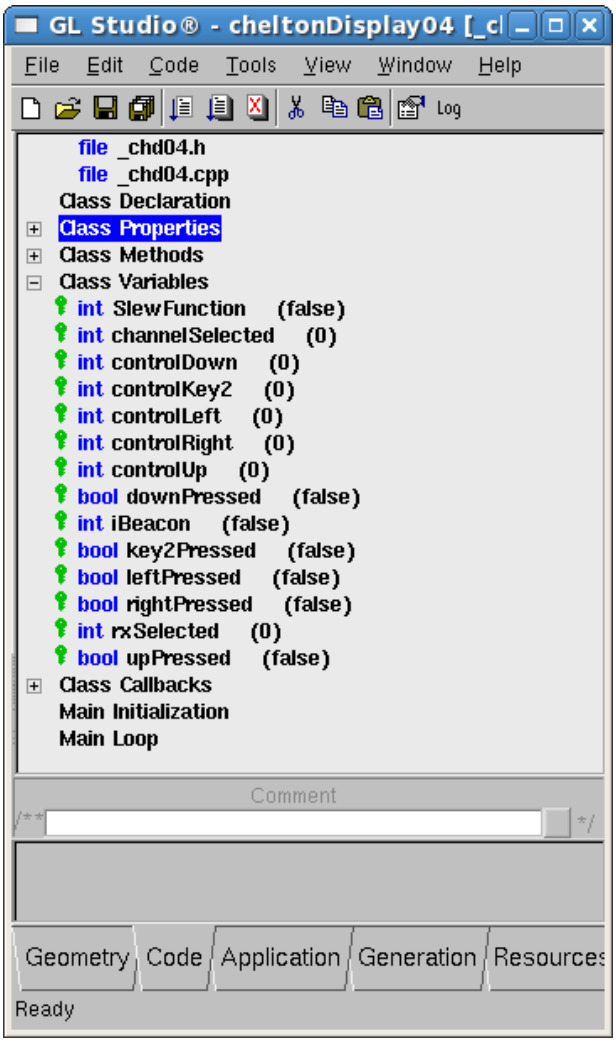


Figura 2.13: *Variables de la pantalla 'HMG Screen'*

Lista de variables utilizadas en la pantalla DF:

- *SlewFunction*: representa si está, o no, activo, el modo Slew.
- *channelSelected:* almacena el número de receptor que ha seleccionado el usuario mediante los cursores.
- *controlDown, controlUp, controlRight, controlLeft, controlKey3:* son variables de control de botón (botones de arriba, abajo, izquierda, derecha y 3), con ellas, se consigue controlar que el usuario pulsa un botón, una vez, a pesar de que lo mantenga pulsado.
- *downPressed, upPressed, rightPressed, leftPressed, key3Pressed:* son variables que indican si están o no pulsados los 5 botones correspondientes.
- *iBeacon:* almacena el número de baliza que se ha detectado por el receptor seleccionado.
- *rxSelected*: almacena el número de receptor seleccionado por el usuario (entre 1 y 4).

Métodos utilizados

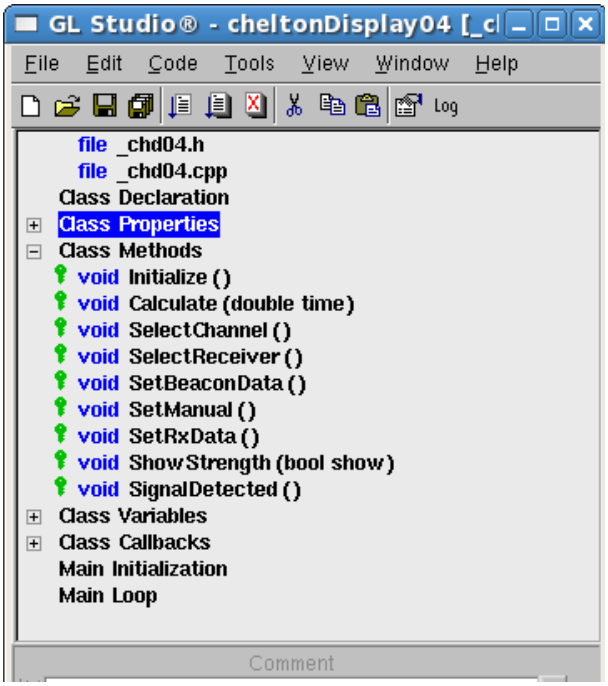


Figura 2.14: *Métodos de la pantalla 'HMG Screen'*

Lista de métodos creados para el funcionamiento de la pantalla DF:

- *SelectChannel*: método mediante el cual, a través de de la detección de pulsaciones de los botones *Izquierda* y *Derecha*, si nos encontramos en el receptor 1 (Manual), variará entre los canales 0 y 9.

- *SelectReceiver*: método mediante el cual, a través de de la detección de pulsaciones de los botones *Arriba* y *Abajo*, cambiaremos de receptor seleccionado entre 1 y 4, para mostrar la información de las balizas que detecta cada uno.

- *SetBeaconData:* método que muestra por pantalla la información de las balizas que son detectadas por un receptor.

- *SetManual:* método que muestra por pantalla el ángulo de rumbo del helicóptero, cuando es editado manualmente por el piloto, activando al función *Slew.*

- *SetRxData*: método que selecciona, si en la pantalla, en el campo *fieldCH*, se debe mostrar el canal seleccionado ('0' … '9') en el caso de que el receptor seleccionado sea el 1, o dos guiones en cualquier otro caso ('- -').

- *ShowStrength:* método que realiza los cálculos necesarios para mostrar la barra de fuerza de señal de la baliza detectada, a la altura que debe estar (apoyado en los datos de latitud y longitud del helicóptero y la baliza, respectivamente).

- *SignalDetected:* método encargado de seleccionar en los distintos switches de señal, de la parte inferior de la pantalla, los receptores que detectan algún tipo de señal. En

caso de ser detectada, se mostrará el receptor en modo inverso (número en negro, sobre fondo amarillo).

Pantalla 5 – MULTI BEACON SCREEN

Esta pantalla muestra el rumbo relativo de hasta 3 balizas *SARBE-7* en los círculos concéntricos, el indicador de posición más sólido, es el que recibe con más fuerza la señal de la baliza.

A los detalles de cada baliza específica (ángulo de rumbo numérico, fuerza de la señal y edad) podemos acceder mediante los botones adyacentes a los números *1, 2* y , *3* en la parte derecha de la pantalla. La baliza uno corresponde con el círculo más alejado. (NOTA no es necesario que la baliza que posee mayor fuerza, sea la mostrada en la circunferencia más exterior).

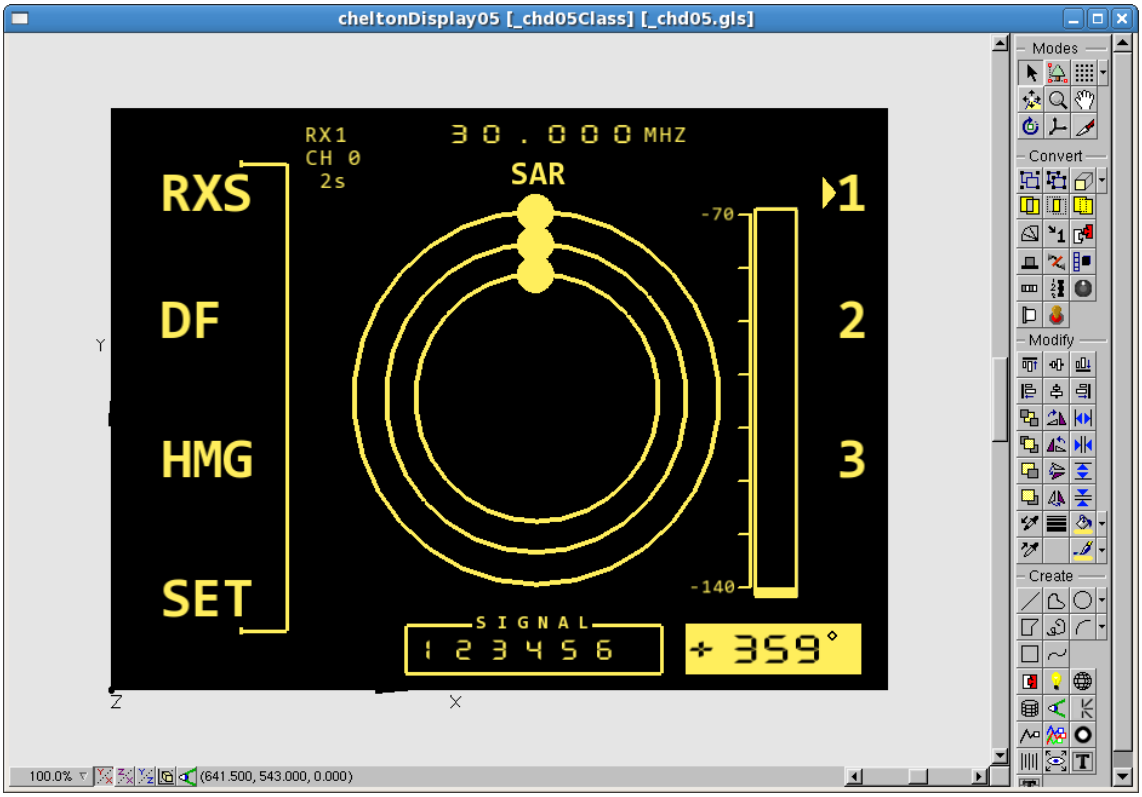


Figura 2.15: *Imagen de la pantalla 'MBC Screen'*

Geometría utilizada

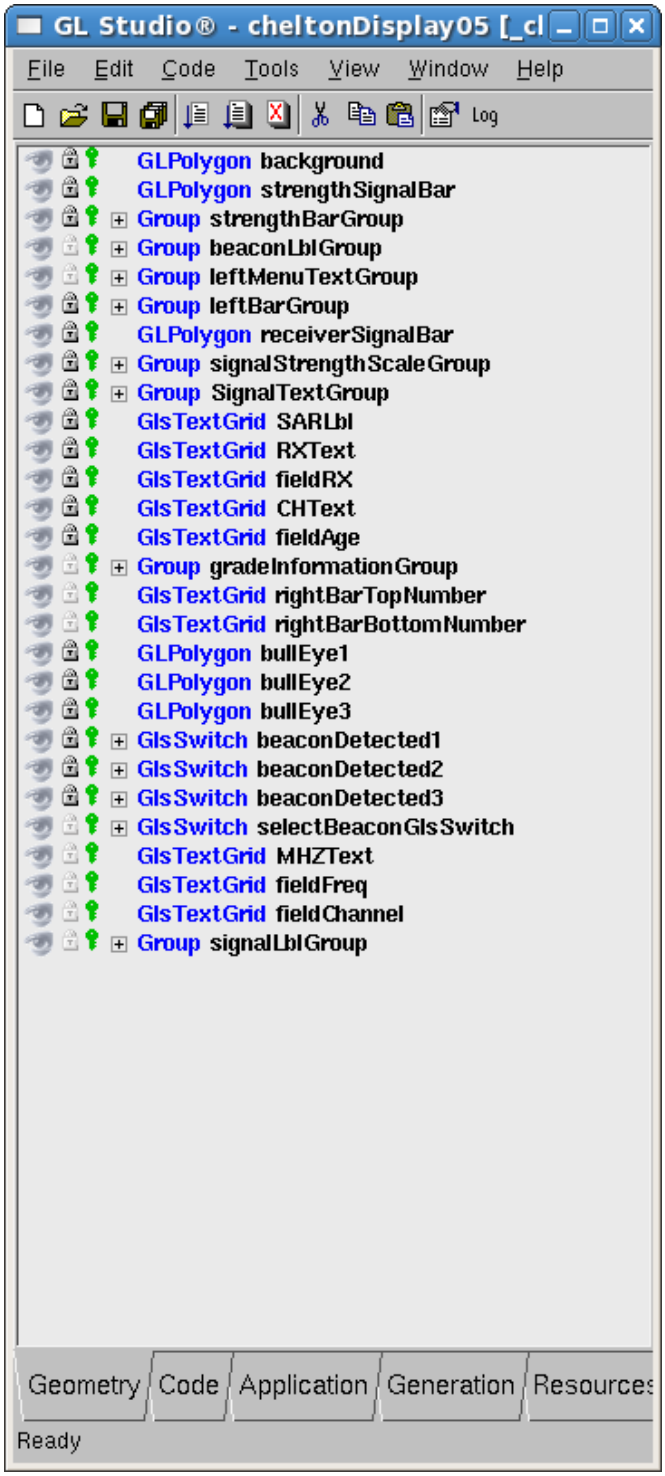


Figura 2.16: *Geometría de la pantalla 'MBC Screen'*

Tras la realización de la imagen completa de la pantalla 3, los elementos creados son:

- Grupo *TOP*: **MHz,SAR, RX y CH**, junto con sus campos variables que representan el valor de los mismos, *fieldGroup*.

- Grupo *CENTER*: agrupa todos los elementos que representan las circunferencias de las 3 posibles balizas detectadas.

- Grupo *LEFT*: agrupa el switch de selección de baliza entre 1 y 3.

- Grupo *DOWN*: agrupa todos los elementos que representan los receptores.

- Grupo *RIGHT:* agrupa los polígonos que representan todo lo relacionado con la barra de fuerza de señal.

- Grupo *fieldGroup*: agrupa todos los campos variables, relacionados con los receptores, frecuencias o canales seleccionados, además del campo edad.

- Grupo *bearingGroup*: agrupa todos los elementos relacionados con el ángulo del rumbo del helicóptero con respecto a la baliza.

Variables utilizadas

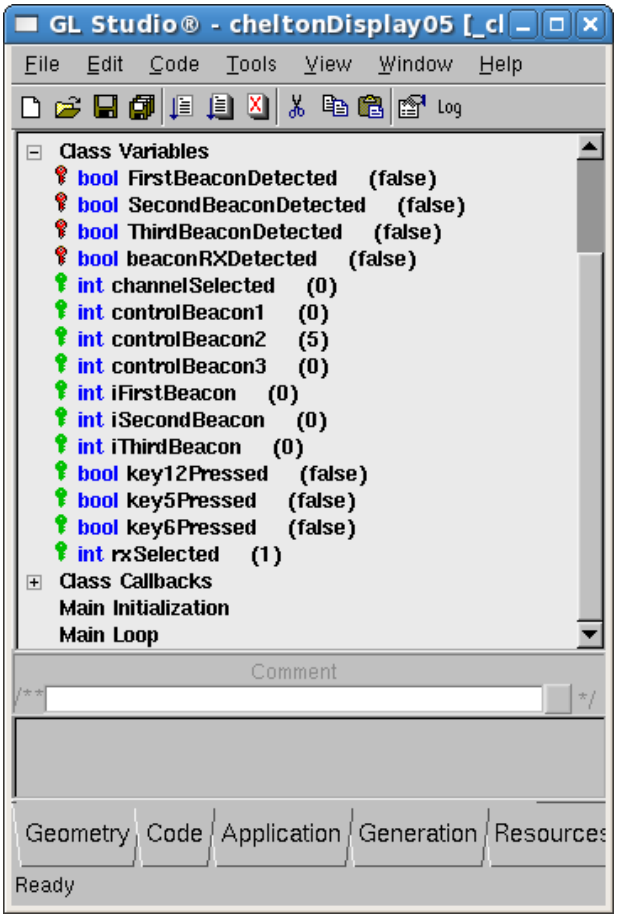


Figura 2.17: *Variables de la pantalla 'MBC Screen'*

Lista de variables utilizadas en la pantalla DF:

- *FirstBeaconDetected, SecondBeaconDetected, ThirdBeaconDetected:* variables booleanas que toman el valor *true* cuando la baliza correspondiente es detectada, o *false* en caso contrario.

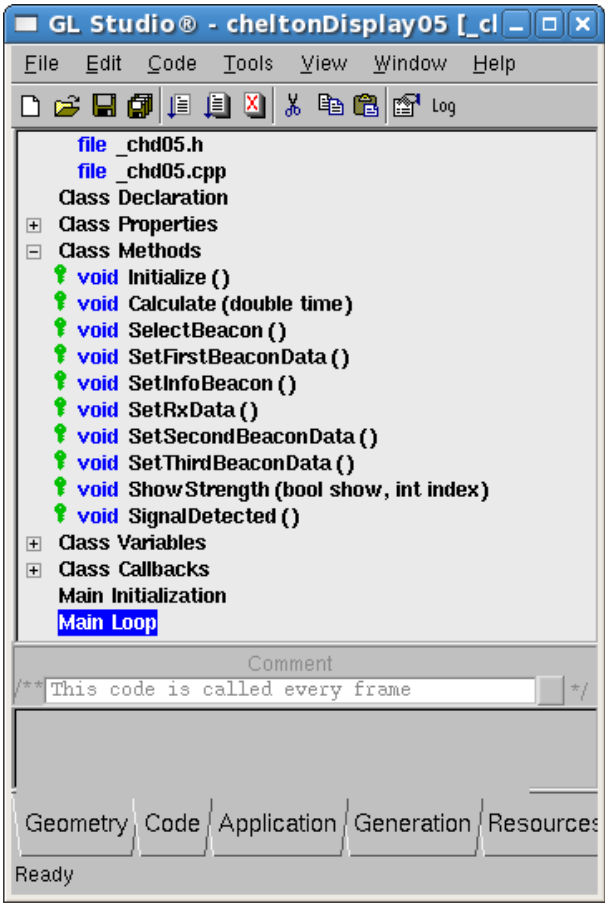
- *beaconRXDetected:* almacena el valor del receptor que detecta la baliza.

- *controlBeacon1, controlBeacon2, controlBeacon3:* variables de control, que almacenan.

- *iFirstBeacon, iSecondBeacon, iThirdBeacon:* variables que almacenan el índice de las 3 balizas detectadas.

- *key12pressed, key5Pressed, key6Pressed:* variables que indican si están o no pulsados los 3 botones correspondientes.

Métodos utilizados



Lista de métodos creados para el funcionamiento de la pantalla DF:

- *SelectBeacon:* método que marca la baliza que selecciona el usuario mediante los botones de la derecha (5,6 y 12)

- *SetFirstBeaconData, SetSecondBeaconData, SetThirdBeaconData:* métodos que muestra por pantalla, en el gráfico, la posición de cada una de las balizas detectadas.

- *SetInfoBeacon:* método que muestra en pantalla la información de la baliza seleccionada por el usuario.

Figura 2.18: *Métodos de la pantalla 'MBC Screen'*

- *ShowStrength:* método que realiza los cálculos necesarios para mostrar la barra de fuerza de señal de la baliza detectada, a la altura que debe estar (apoyado en los datos de latitud y longitud del helicóptero y la baliza, respectivamente).

- *SignalDetected:* método encargado de seleccionar en los distintos switches de señal, de la parte inferior de la pantalla, los receptores que detectan algún tipo de señal. En caso de ser detectada, se mostrará el receptor en modo inverso (número en negro, sobre fondo amarillo).

Pantalla 6 – MESSAGES SCREEN

Esta pantalla muestra el número de mensajes que están almacenados, tanto en modo *DSC* como en modo *COSPAS / SARSAT* y además indica que uno o más mensajes todavía no han sido leídos con el símbolo ‘’. Si el número de mensajes no es *cero* entonces el contenido del mensaje puede ser visto pulsando en las teclas de *DSC* y *SARSAT*

Figura 2.19: *Imagen de la pantalla ‘MSG Screen’*

Geometría utilizada

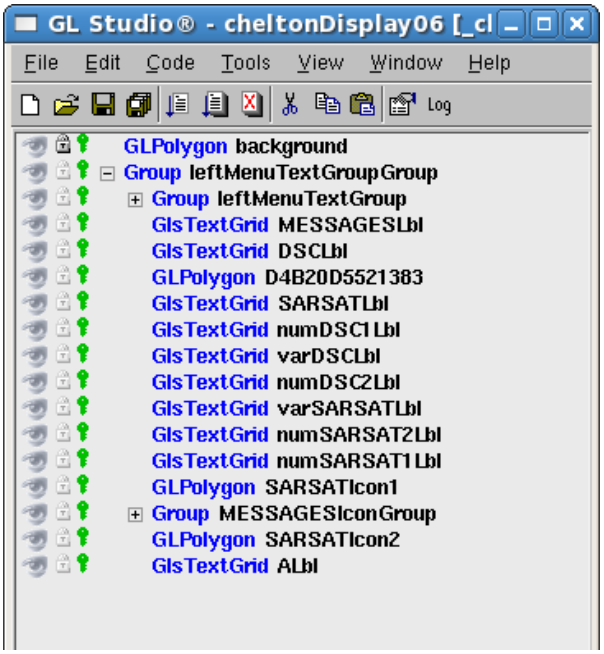


Figura 2.20: *Geometría de la pantalla 'MSG Screen'*

Tras la realización de la imagen completa de la pantalla 6, los elementos creados son:

- Grupo *TOP*: **MHz,SAR, RX y CH**, junto con sus campos variables que representan el valor de los mismos, *fieldGroup*.
- Grupo *CENTER*: agrupa todos los elementos que representan las circunferencias de las 3 posibles balizas detectadas.
- Grupo *LEFT*: agrupa el switch de selección de baliza entre 1 y 3.
- Grupo *DOWN*: agrupa todos los elementos que representan los receptores.
- Grupo *RIGHT:* agrupa los polígonos que representan todo lo relacionado con la barra de fuerza de señal.

Variables y métodos utilizados

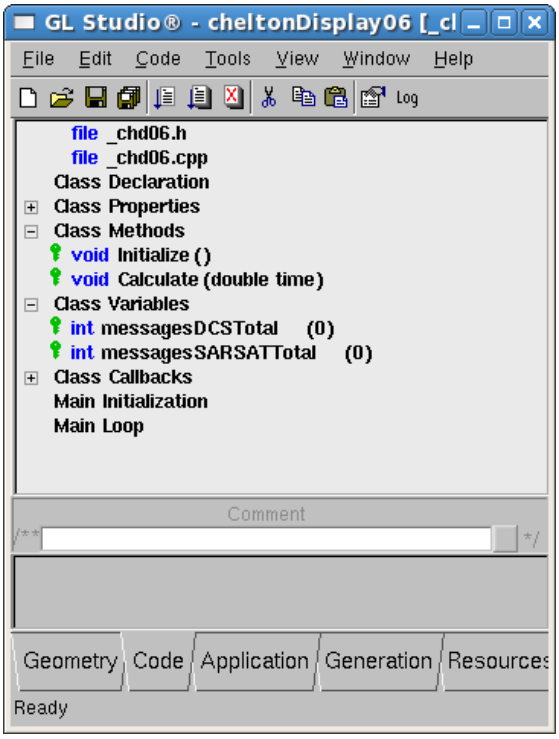


Figura 2.21: *Variables de la pantalla 'MSG Screen'*

Las variables que contiene esta pantalla son 2:

- messagesDSCTotal: almacena el número de mensajes que se reciben en el receptor DSC.
- messagesSARSATTotal: almacena el número de mensajes que se reciben en el receptor SARSAT.

Pantalla 7 – COSPAS / SARSAT MESSAGE SCREEN

Accediendo a esta pantalla, podemos ver el último mensaje recibido en modo *SARSAT*. Otros mensajes, si hay, pueden ser seleccionado, utilizando las teclas de *arriba*|*abajo* y de *izquierda*|*derecha*.

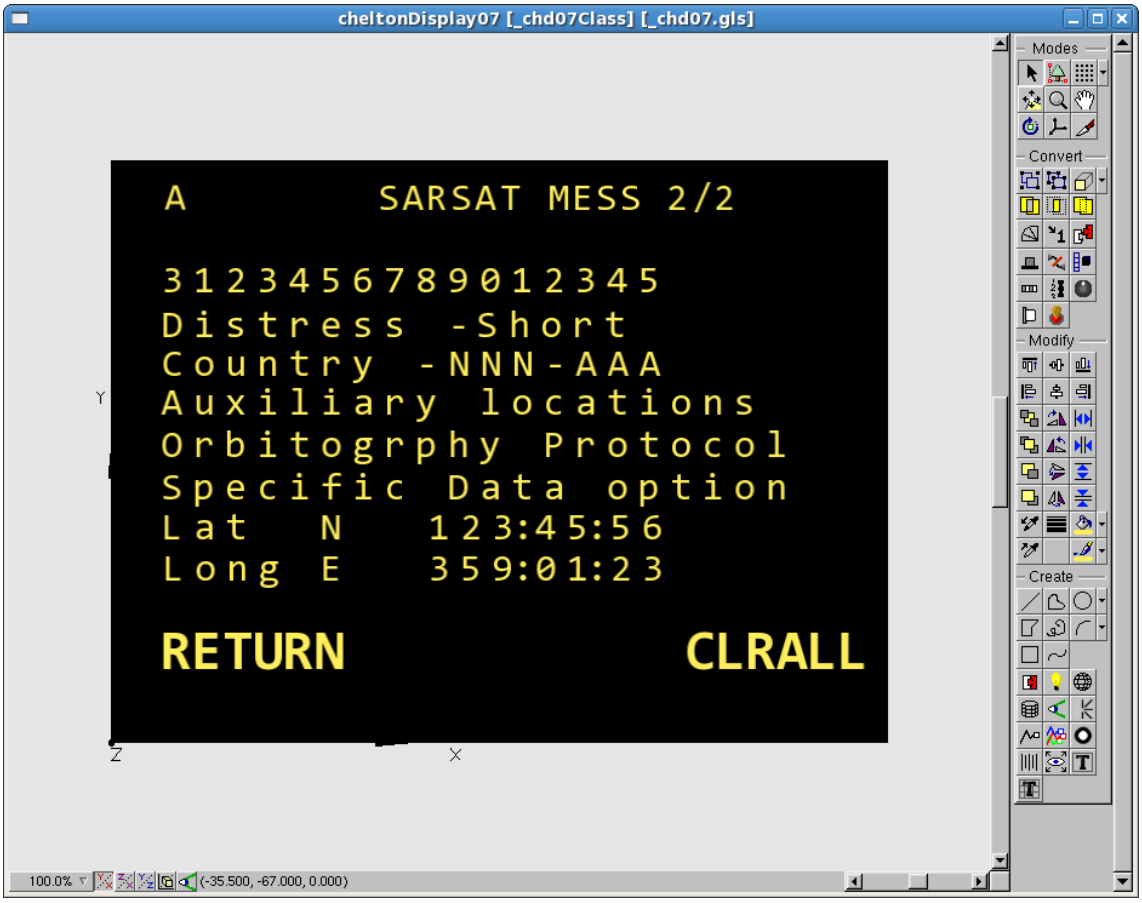
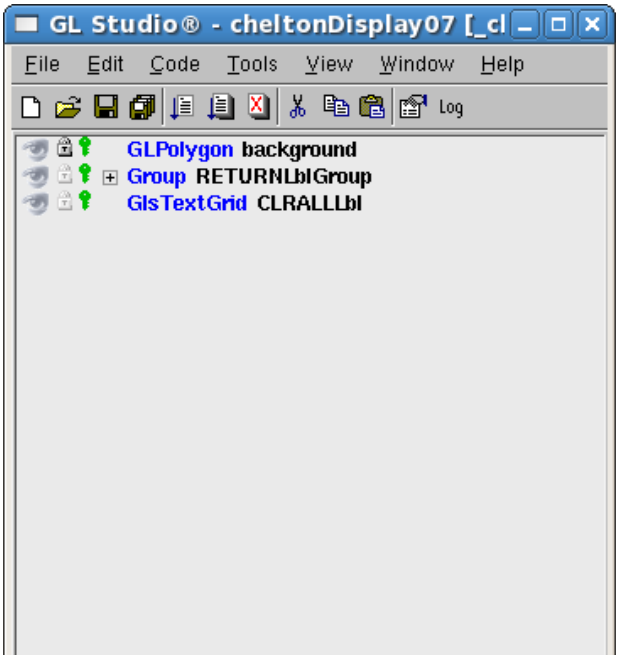


Figura 2.22: *Imagen de la pantalla 'SARSAT Screen'*

La pantalla en esta implementación muestra los siguientes datos:

- Número de mensaje que se está leyendo y número total de mensajes.
- Datos Raw SARSAT, 16 caracteres numéricos.
- La naturaleza del mensaje (*TEST* o *DISTRESS*).
- Información del país.
- Localización auxiliar del dispositivo.
- Protocolo utilizado.
- Las coordenadas de latitud y longitud para las diferentes balizas que el GPS facilita. (Si la baliza no da ningún tipo de información GPS, es decir, latitud y longitud, se mostrará en pantalla XXXXX.

Geometría utilizada



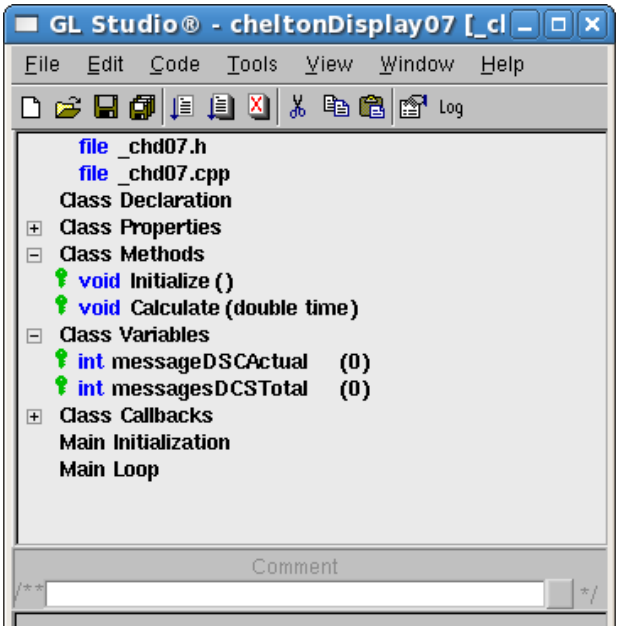
Tras la realización de la imagen completa de la pantalla 7, los elementos creados son:

- Grupo *RETURNLblGroup:* Grupo que contiene todas las etiquetas de texto en las que se muestra la información de la pantalla.

- Grupo *CENTER*: agrupa todos los elementos que representan las circunferencias de las 3 posibles balizas detectadas.

Figura 2.23: *Geometría de la pantalla 'SARSAT Screen'*

Variables



Las variables que contiene esta pantalla son 2:

- messageDSCActual almacena el número del mensaje que leemos actualmente.

- messagesDSCTotal: almacena el número de mensajes recibidos en total.

Figura 2.24: *Variables y métodos de la pantalla 'SARSAT Screen'*

Pantalla 8 – MARITIME DSC MESSSAGE SCREEN

Accediendo a la pantalla de *Maritime DSC Messages*, la pantalla mostrará el último mensaje recibido en pantalla. Otros mensajes DSC, si existieran, pueden ser seleccionando mediante los botones *arriba* | *abajo* e *izquierda* | *derecha*.

Los mensajes DSC pueden ser borrados pulsando la tecla *CLRALL*.

El icono ‘ ‘ es utilizado para indicar que ha mensajes sin leer (no han sido vistos todavía).

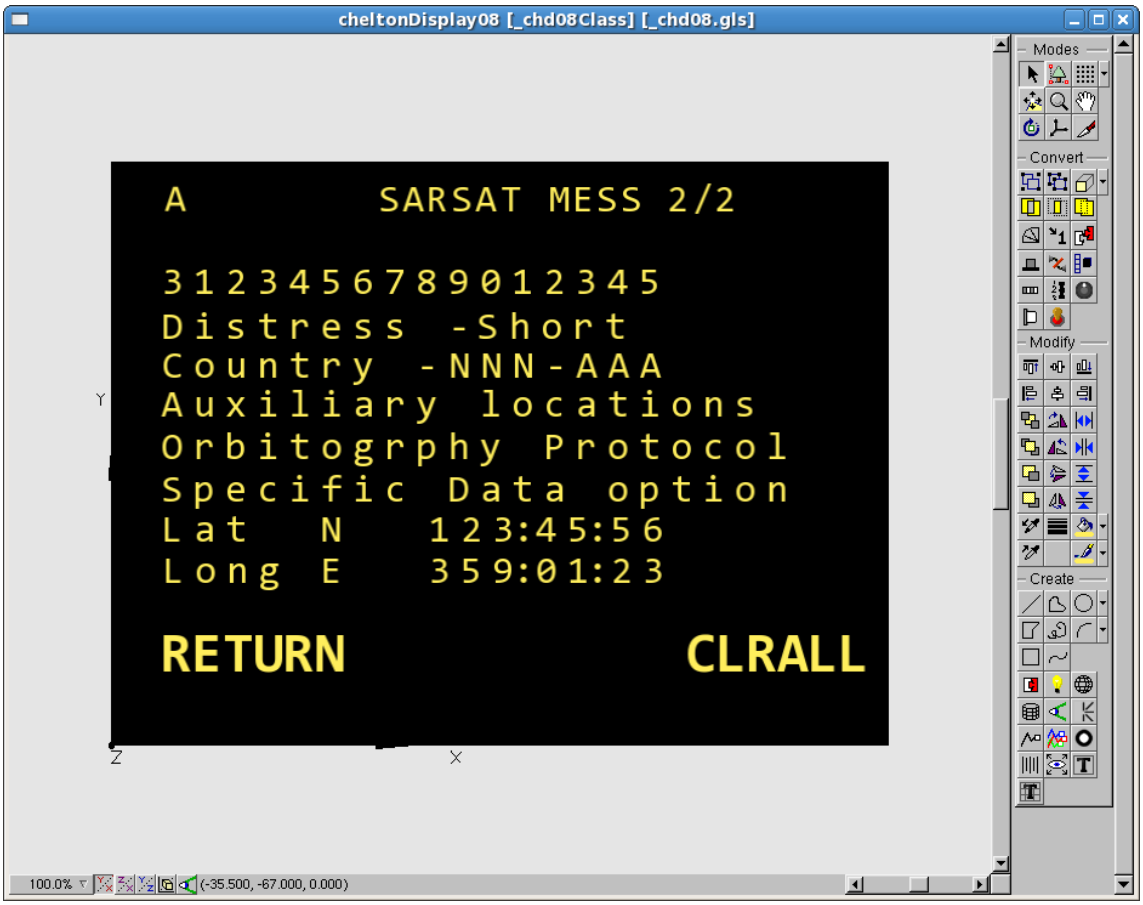


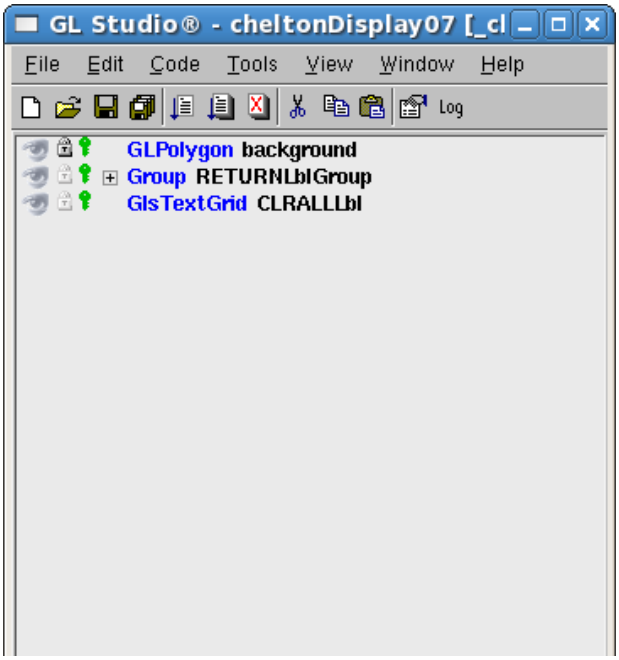
Figura 2.25: *Imagen de la pantalla ‘DSC Screen’*

Este tipo de mensajes, muestra la siguiente información:

- Número de mensaje que se está leyendo y número total de mensajes.
- FOR: Formato del mensaje recibido.
- CAT: *Category Identifier* Identificador de categoría. Puede ser una de las siguientes:
  - 100: Rutina
  - 110: Urgencia

    - 112: Desastre
- CID: *Caller Identifier* Identificador de llamada. Es de tipo numérico y representa la dirección del mensaje (*MMSI*).
- SID: *Self Identifier*. Es de tipo numérico y representa la dirección del emisor que envía el mensaje (*MMSI*).
- DID: Distressed Identifier. Identificador numérico que muestra la dirección (*MMSI*) del desastre.
- NOD: *Nature of Distress.* Este es un número que refleja el desastre sucedido (por ejemplo, 107 = *Sinking*).
- SUB: *Subsequent Communications type*. Tipo de Comunicación de subsecuencia. Este indica el significado por el cual se comunica con el mensaje el emisor.
- TC1 y TC2: *Tele-Commands* 1 y 2.
- LAT | LON: Coordenadas enviadas por el emisor.
- TIM: Edad del mensaje en el aparato (desde cuándo llegó).
- CSR: *Called station, received channel or frequency.*
- CST: *Called station, transmit channel or frequency.*

Geometría utilizada



Tras la realización de la imagen completa de la pantalla 8, los elementos creados son:

- Grupo *RETURNLblGroup:* Grupo que contiene todas las etiquetas de texto en las que se muestra la información de la pantalla.

- Grupo *CENTER*: agrupa todos los elementos que representan las circunferencias de las 3 posibles balizas detectadas.

Figura 2.26: *Geometría de la pantalla 'DSC Screen'*

Variables

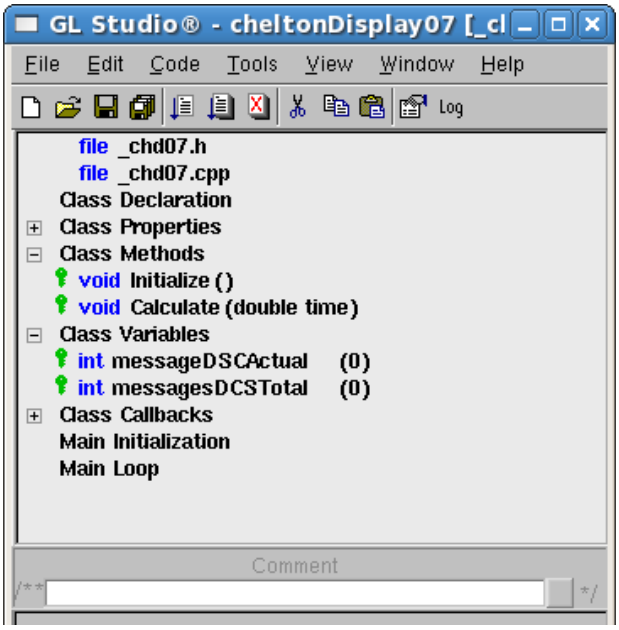


Figura 2.27: *Variables y métodos de la pantalla 'DSC Screen'*

Las variables que contiene esta pantalla son 2:

- messageDSCActual
  almacena el número del mensaje que leemos actualmente.

- messagesDSCTotal:
  almacena el número de mensajes recibidos en total.

Pantalla 9 – BUILT IN TEST SCREENS (BITE)

Para poder acceder a esta pantalla *BITE* , debemos entrar desde la pantalla 2 *RXR STATUS* presionando el botón *BIT*. Esta pantalla muestra, en términos generales, el estado del sistema de *Controller / DF* (*General, Direction Finder y Controller*). Una vez iniciado el test, puede repetirse tantas veces como uno quiera, pulsando la tecla *TST*.

Los resultados detallados de los fallos para cada una de las categorías pueden ser mostradas seleccionando la categoría, con los botones de *arriba* | *abajo* o el potenciómetro y posteriormente pulsando la tecla *REP (Report)*.

Figura 2.28: *Imagen de la pantalla 'BITE Screen'*

Geometría utilizada

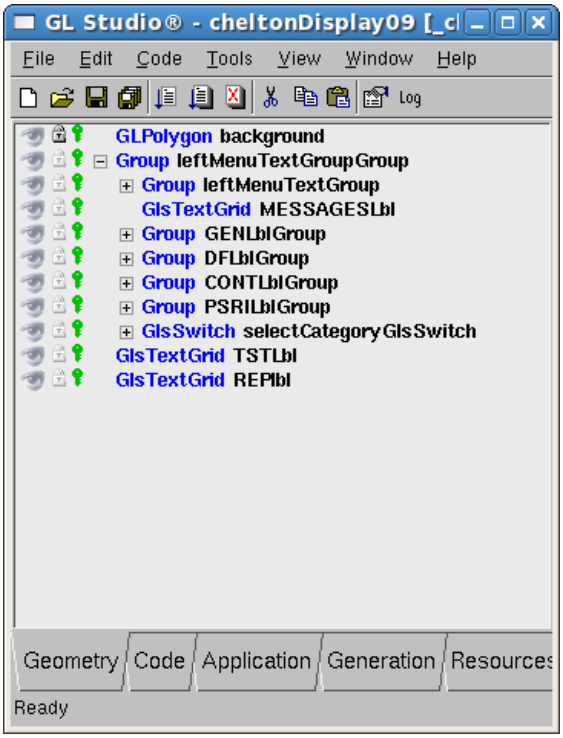


Figura 2.29: *Geometría de la pantalla 'BITE Screen'*

Tras la realización de la imagen completa de la pantalla 9, los elementos creados son:

- Grupo *leftMenuTextGroup:* Contiene todas las etiquetas del menú lateral izquierdo.
- Grupo *GENLblGroup, DFLblGroup, ContLblGroup, PSRILblGroup*: Cada uno de ellos contiene las dos etiquetas de texto correspondientes a su nombre.
- *TSTLbl y RepLbl*: Etiquetas de los textos de la derecha.

Variables utilizadas

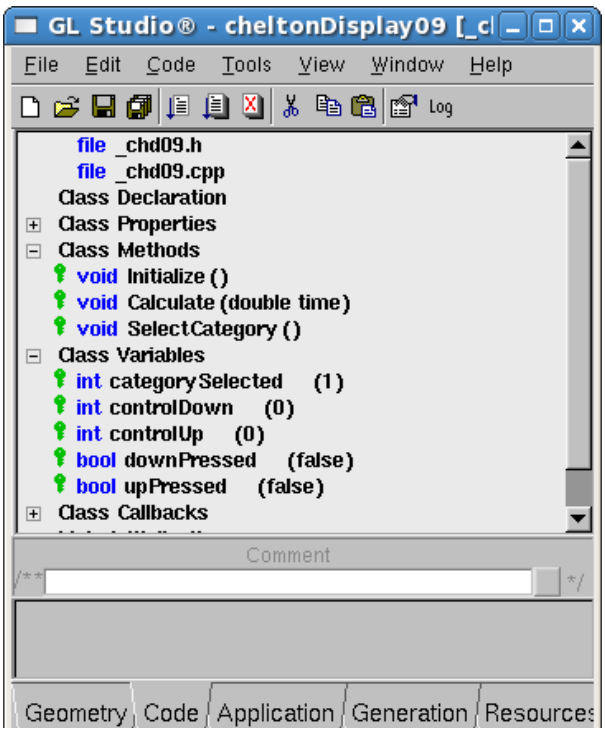


Figura 2.30: *Variables y métodos de la pantalla 'BITE Screen'*

Las varirables utilizadas en la pantalla 9 son las siguientes:

- *categorySelected*: Almacena la posición del switch de selección, de forma que, dependiendo del valor que posea, el menú de report será Gen Report, DF Report, Controller Report o PSRI Report.

- *controlDown, controlU :* Variables para controlar que pulsamos una sola vez la tecla de *abajo* o *arriba*.

- *downPressed,upPressed*: Variables que almacenan si hemos pulsado la tecla *abajo* o *arriba*.

Pantalla 10 – GENERAL REPORT SCREEN

Esta pantalla contiene los indicadores de fallos para las comunicaciones entre el controlador y el DF y el controlador de forma interna.

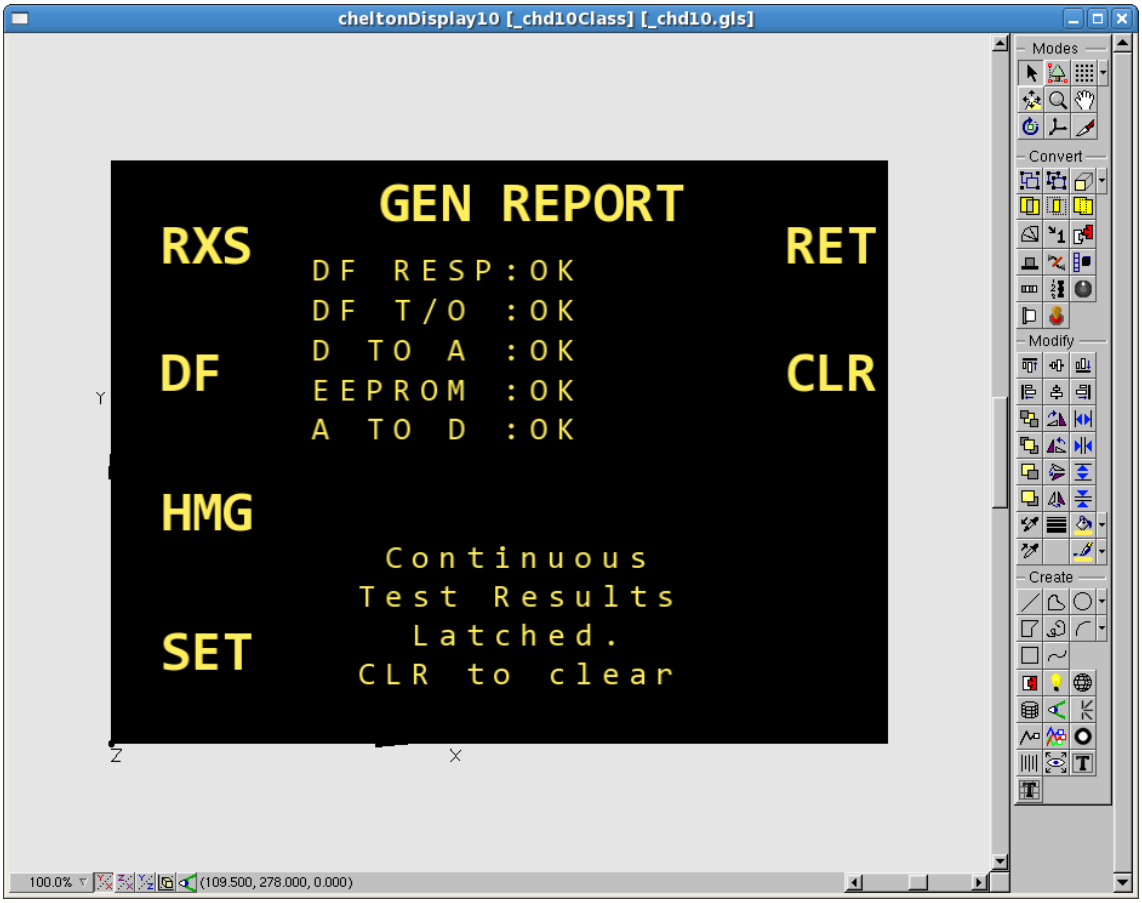


Figura 2.31: *Imagen de la pantalla 'GEN Report Screen'*

Geometría Utilizada.

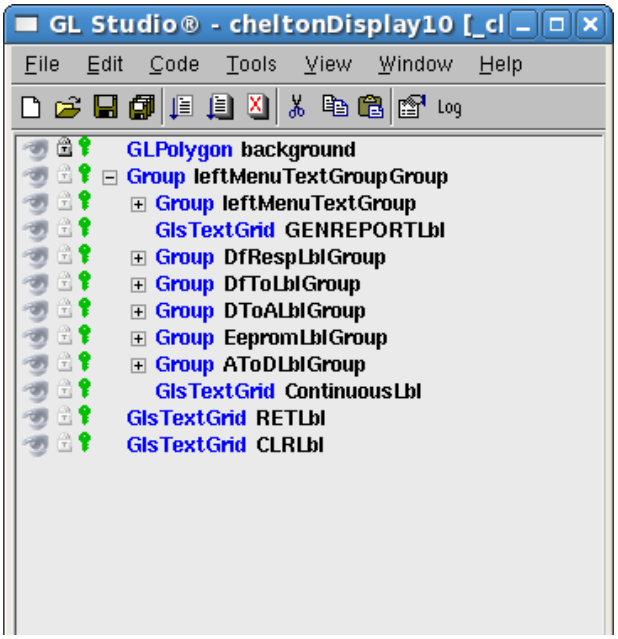


Figura 2.32: *Geometría de la pantalla 'GEN Report Screen'*

Tras la realización de la imagen completa de la pantalla 10, los elementos creados son:

- Grupo *leftMenuTextGroup:* Grupo que contiene todo tel texto del menú de la izquierda.
- Grupo *DfRespLblGroup:* Contiene las etiquetas del primer ítem.
- Grupo *DToALblGroup:* Contiene las etiquetas del segundo ítem.
- Grupo *EepromLblGroup*: Contiene las etiquetas del tercer ítem.
- Grupo *AToDLblGroup*: Contiene las etiquetas del cuarto ítem.

Variables utilizadas

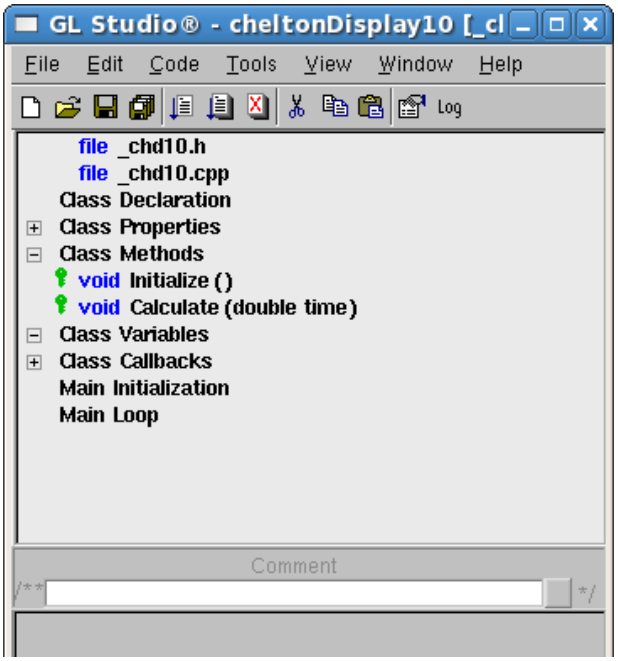


Figura 2.33: *Variables y métodos de la pantalla 'GEN Report Screen'*

Carece de variables y de métodos, debido a que, es una pantalla que únicamente muestra información, no necesita calcular ni generar nada.

Pantalla 11 – DF REPORT SCREEN

Muestra los valores obtenidos tras la prueba de distintos parámetros del *Direction Finder*

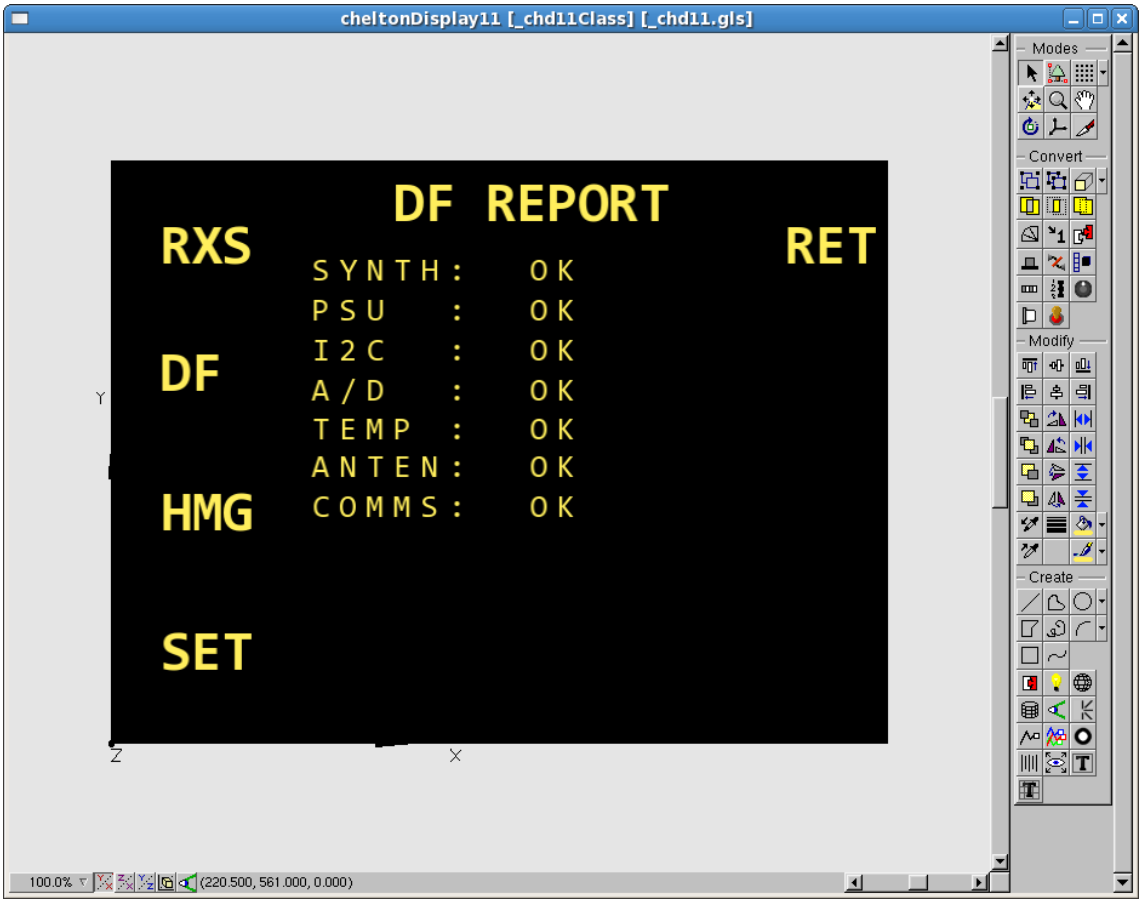


Figura 2.34: *Imagen de la pantalla 'DF Report Screen'*

Geometría Utilizada

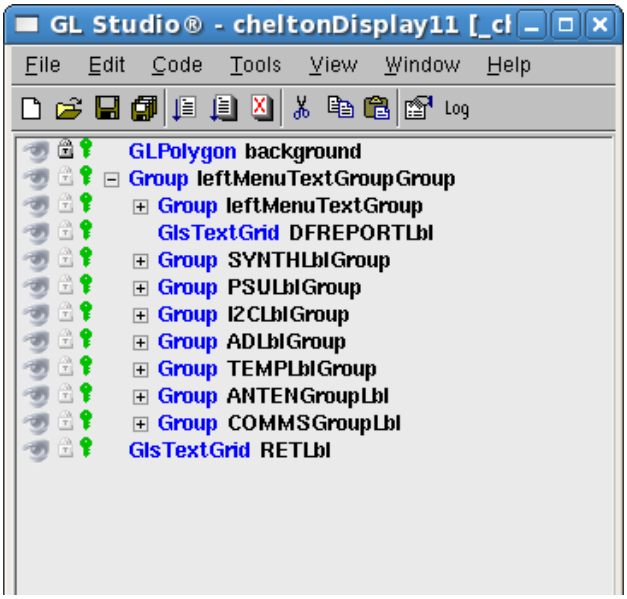


Figura 2.35: *Geometría de la pantalla 'DF Report Screen'*

Tras la realización de la imagen completa de la pantalla 11, los elementos creados son:

- Grupo *leftMenuTextGroup:* Grupo que contiene todo tel texto del menú de la izquierda.

- Grupos *SYNTHLblGroup, PSULblGroup, I2CLblGroup, ADLblGroup, TEMPLblGroup, ANTENGroupLbl* y *COMMSGroupLbl:* Contienen las etiquetas de los siete ítems correspondientes a su nombre.

Variables y métodos

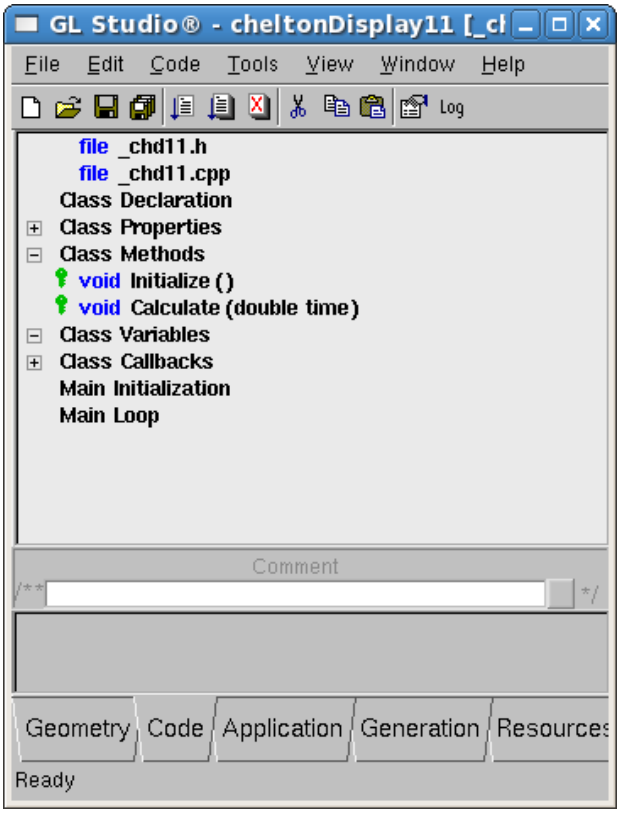


Figura 2.36: *Variables y métodos de la pantalla 'DF Report Screen'*

Carece de variables y de métodos, debido a que, es una pantalla que únicamente muestra información, no necesita calcular ni generar nada.

Pantalla 12 – CONTROLLER REPORT SCREEN

Muestra los valores obtenidos tras la prueba de distintos parámetros del *Controller*

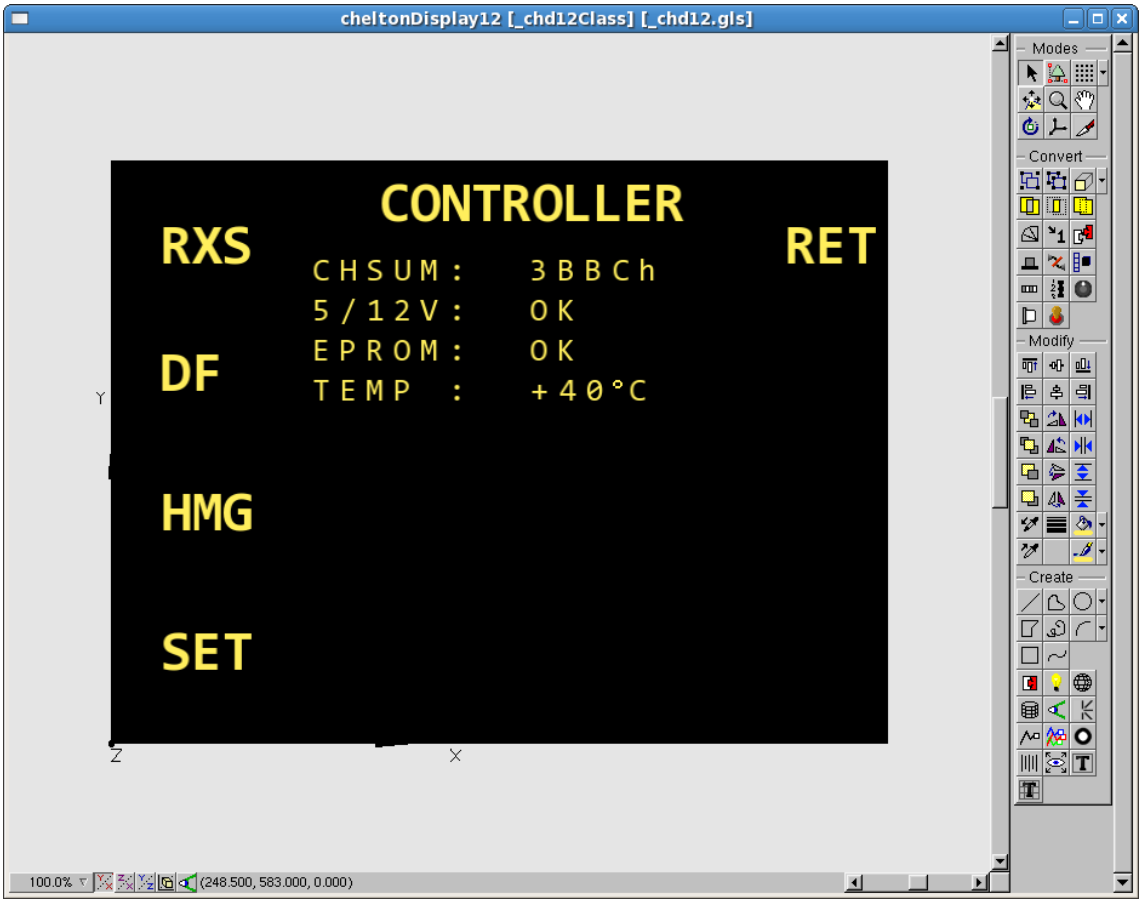


Figura 2.37: *Imagen de la pantalla 'CONTROLLER Screen'*

Geometría Utilizada

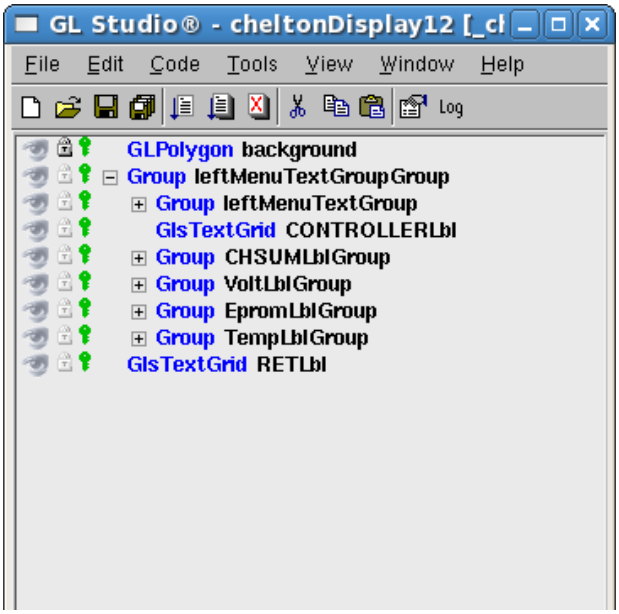


Figura 2.38: *Geometría de la pantalla 'CONTROLLER Screen'*

Tras la realización de la imagen completa de la pantalla 12, los elementos creados son:

- Grupo *leftMenuTextGroup:* Grupo que contiene todo tel texto del menú de la izquierda.

- Grupos *CHSUMLblGroup, VoltLblGroup, EpromLblGroup, TempLblGroup:* Contienen las etiquetas de los siete ítems correspondientes a su nombre.

Variables Utilizadas

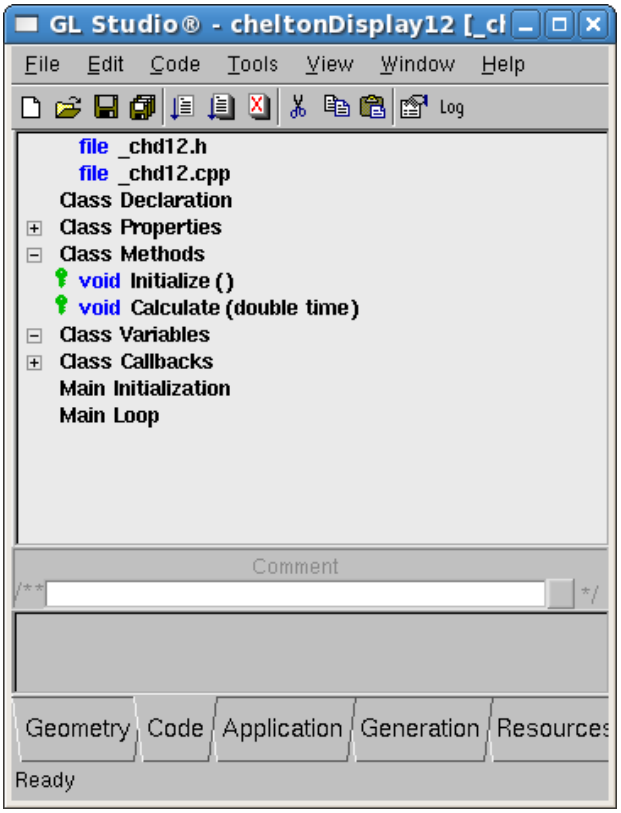


Figura 2.39: *Variables y métodos de la pantalla 'CONTROLLER Screen'*

Carece de variables y de métodos, debido a que, es una pantalla que únicamente muestra información, no necesita calcular ni generar nada.

Pantalla 13 – PSR REPORT SCREEN

Esta pantalla muestra los fallos obtenidos para el interrogador *PSR*.

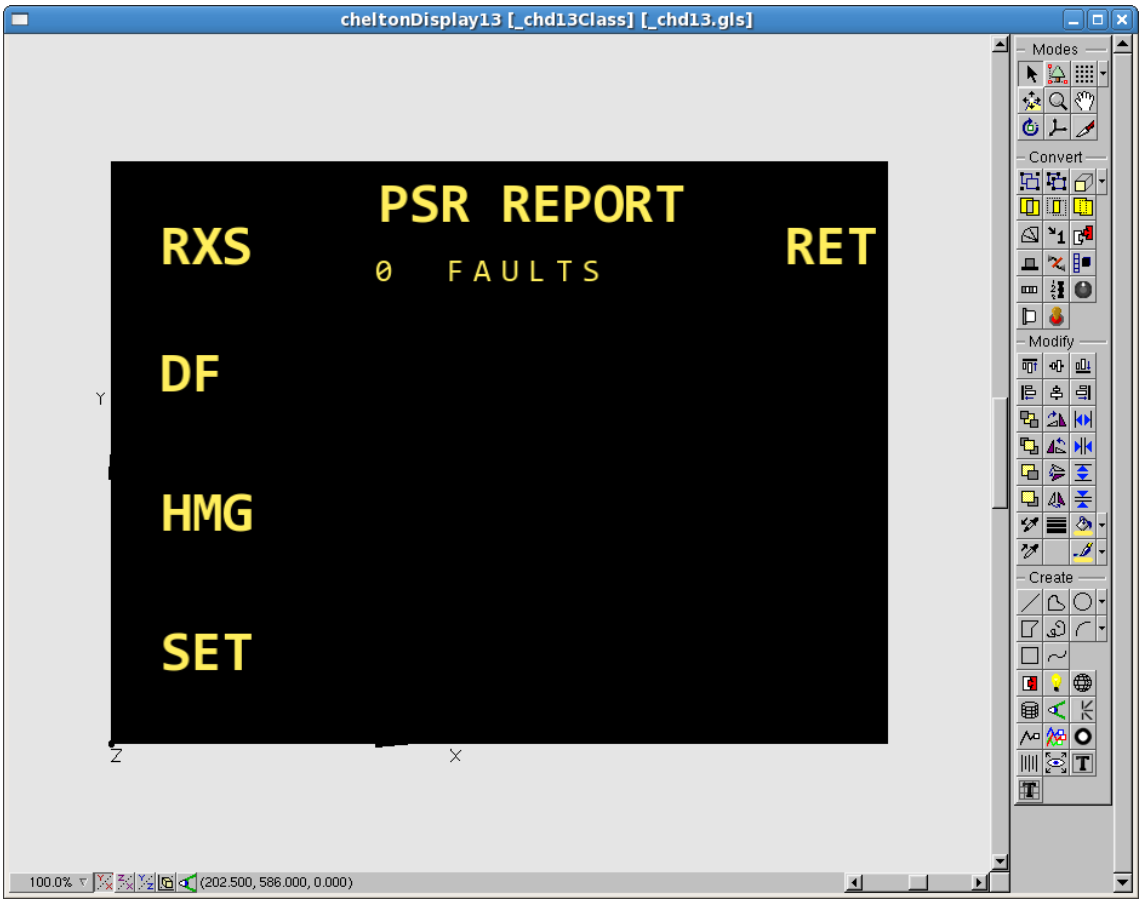


Figura 2.40: *Imagen de la pantalla 'PSR Report Screen'*

Carece de variables y métodos, debido a que, en esta pantalla no se realiza ninguna función.

Pantalla 14 – RECEIVER PROGRAMMING SCREEN

A esta pantalla accedemos desde la pantalla 2 *RXR STATUS* presionando el botón *PRG*. Esta pantalla muestra la información asociada con cada uno de los receptores, mediante los cursores, podemos seleccionar un receptor u otro y cambiar así sus características.

En ella podemos seleccionar, el canal de cada receptor que queremos tener activado. En el caso del receptor 1, elegiremos entre los 10 canales posibles, en los receptores 2,3 y 4, elegiremos entre el principal y el auxiliar.

Podremos activar o desactivar el receptor y cambiar otro tipo de valores como por ejemplo, activar o desactivar el sonido.

En todo caso, para poder editar cualquier parámetro, es necesario que pulsemos la tecla *Enter* y seguidamente, cambiemos el valor mediante las teclas *arriba* | *abajo* e *izquierda* | *derecha*.

Figura 2.41: *Imagen de la pantalla 'PROG RX Screen'*

Geometría utilizada

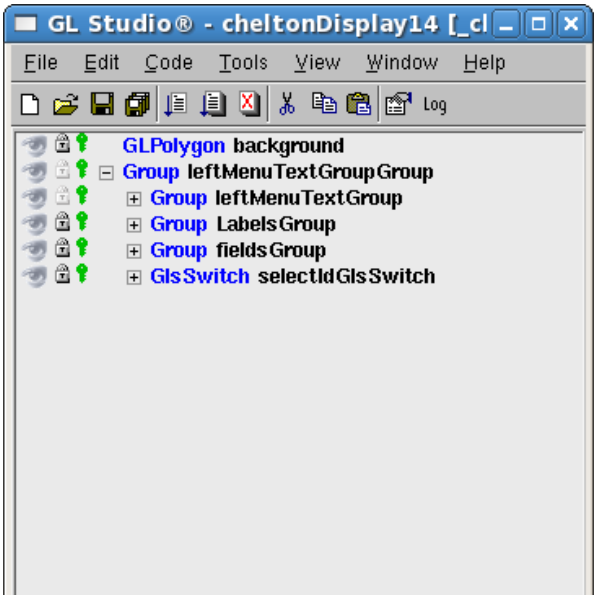


Figura 2.42: *Geometría de la pantalla 'PROG RX Screen'*

Tras la realización de la imagen completa de la pantalla 14, los elementos creados son:

- Grupo *leftMenuTextGroup:* Grupo que contiene todo tel texto del menú de la izquierda.
- Grupo *LabelsGroup:* Contiene todas las etiquetas que no varían (RX, Channel, Freq…).
- Grupo *fieldsGroup*: Contiene todos los campos variables.
- GlsSwitch *selecteIdGlsSwitch:* Switch que contiene posición del cursor.

Variables utilizadas

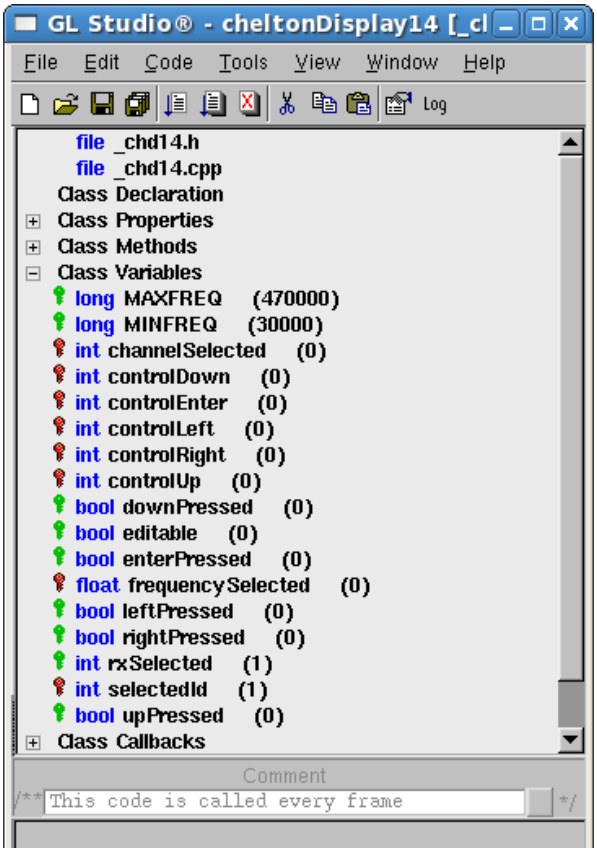


Figura 2.43: *Variables de la pantalla 'PROG RX Screen'*

Variables utilizadas:

- *MAXFREQ,MINFREQ:* Almacenan la frecuencia mácima y mínima que puede almacenar el receptor elegido.
- *rxSelected.channelSelected:* Almacenan el receptor y el canal seleccionado.
- *controlDown, controlEnter, controlLeft, controlRight, controlUp:* Variable para controlar que solo se ha pulsado una vez las distintas teclas.
- *downPressed,enterPressed, leftPressed, rightPressed, upPressed:* Controlan que el botón se ha pulsado.
- *editable:* Variable para controlar si el usuario puede o no editar los campos.

Métodos

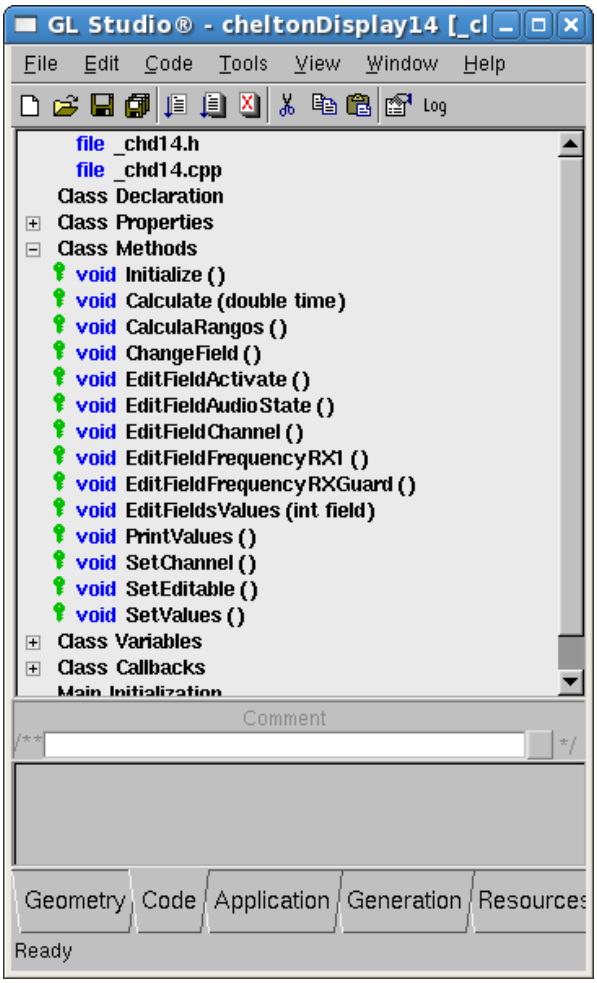


Figura 2.44: *Métodos de la pantalla 'PROG RX Screen'*

Métodos utilizados:

- *CalculaRangos:* Método que proporciona el rango posible para las frecuencias auxiliarse de cada receptor.
- *ChangeField:* Método mediante el cual elegimos el campo que queremos editar.
- *EditFieldActivate, EditFieldAudioState, EditFieldChannel, EditFieldFrequencyRX1, EditFieldFrequencyRXGuard, EditFieldsValues:* Métodos mediante los cuales editamos cada uno de los campos que conllevan sus nombres.
- *SetChannel:* Asigna el canal elegido.
- *SetEditable:* Método que nos permite modificar los campos o no.
- *SetValues:* Método encargado de asignar los diferentes valores.

Pantalla 15 – SETUP SCREEN

Podemos acceder a esta pantalla desde muchas otras pantallas, presionando la tecla *SET*. En esta pantalla se dará acceso a distintas pantallas de *Set-up*, concretamente para configurar las siguientes características:

- Configurar las opciones de *Maritime DSC*, pulsando *SU DSC*.
- Configurar los parámetros de *DF*, pulsando *SU DF.*
- Configurar los parámetros de *CSAR*, pulsando *SU CSAR.*
- Configurar los parámetros de *Controller*, pulsando *SU CONT*.
- Borrar una *DISTRESS ALERT* presionando la tecla *CLR ALERT*.

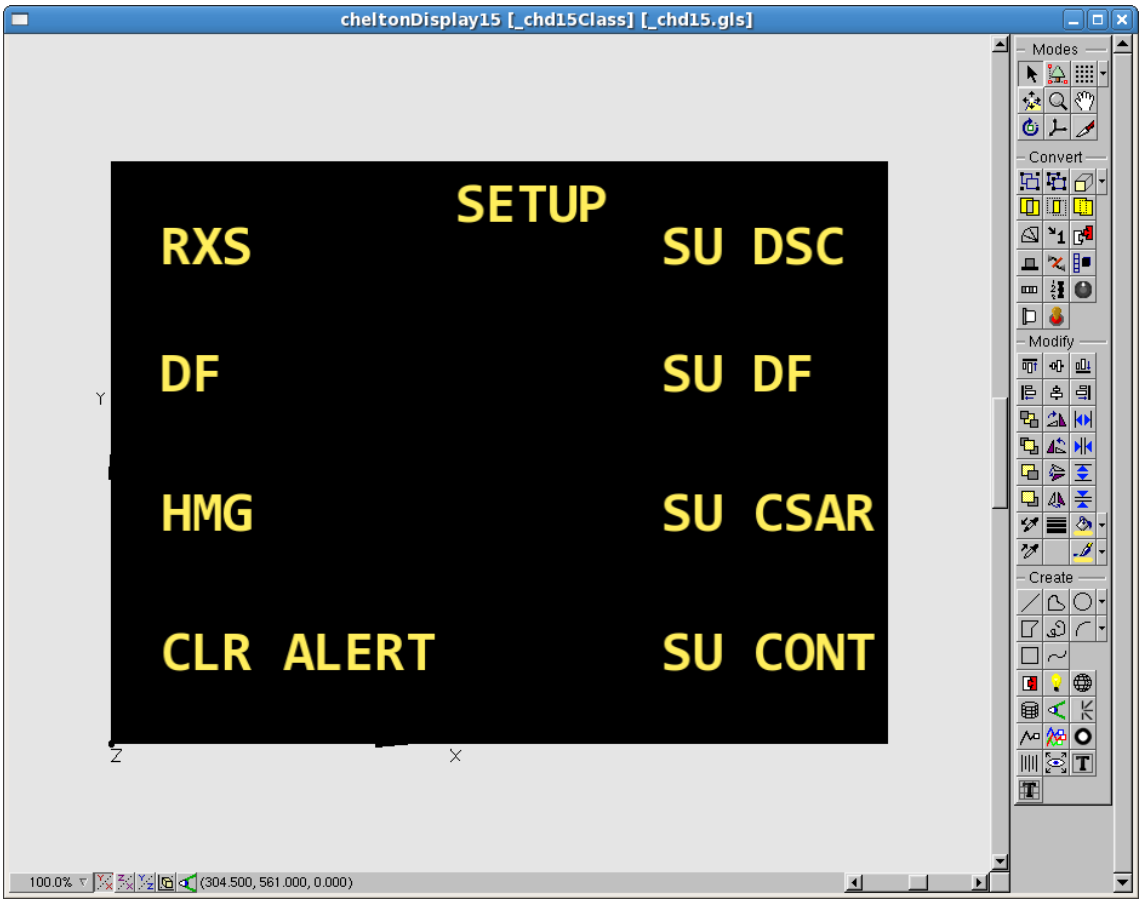


Figura 2.45: *Imagen de la pantalla 'SETUP Screen'*

Geometría utilizada

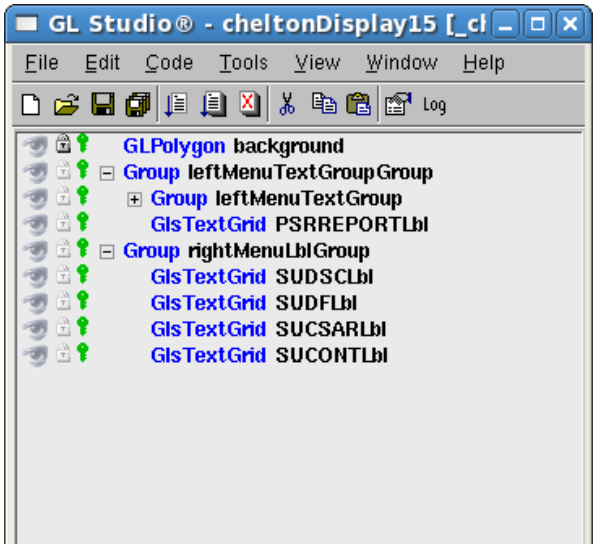


Figura 2.46: *Geometría de la pantalla 'SETUP Screen'*

La geometría utilizada para la pantalla 15 es la siguiente:

- Grupo *leftMenuTextGroup:* Etiquetas del menú lateral izquierdo.

- *SUDSCLbl, SUDFLbl, SUCSARLbl, SUCONTLbl:* Etiquetas de los distintos menús de setup, situados a la derecha de la pantalla.

Variables y métodos utilizados

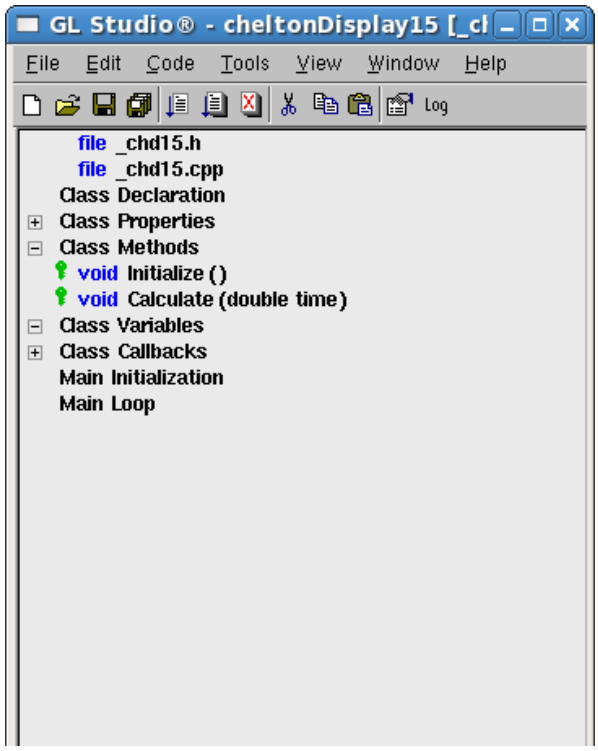


Figura 2.47: *Variables y métodos de la pantalla 'SETUP Screen'*

Carece de variables y de métodos, debido a que, es una pantalla que únicamente muestra información, no necesita calcular ni generar nada.

Pantalla 16 - SU DSC

En esta pantalla, el usuario puede seleccionar qué tipos de mensaje va a poder mostrar el receptor *DF*, de los datos recibidos en el canal *Maritime 70*. Se utiliza para reducir el número de mensajes recibidos. Si un tipo de mensaje está desactivado, el receptor lo rechazará.

Los distintos tipos de mensaje *Maritime* son los siguientes:

- 102: Llamada selectiva a un área geográfica específica.
- 112: Llamada de alerta por desastre (Normalmente está activado siempre).
- 114: Llamada selectiva a un grupo de interés común.
- 116: Todas las Ships Call.
- 120: Llamada selectiva a una estación individual.
- 123: Llamadas de selección automática a una instalación individual.
- DIS: Este *switch* activa o desactiva la decodificación para los mensajes de tipo *DISTRESS*.
- URG: Este *switch* activa o desactiva la decodificación para los mensajes de tipo *URGENCY.*

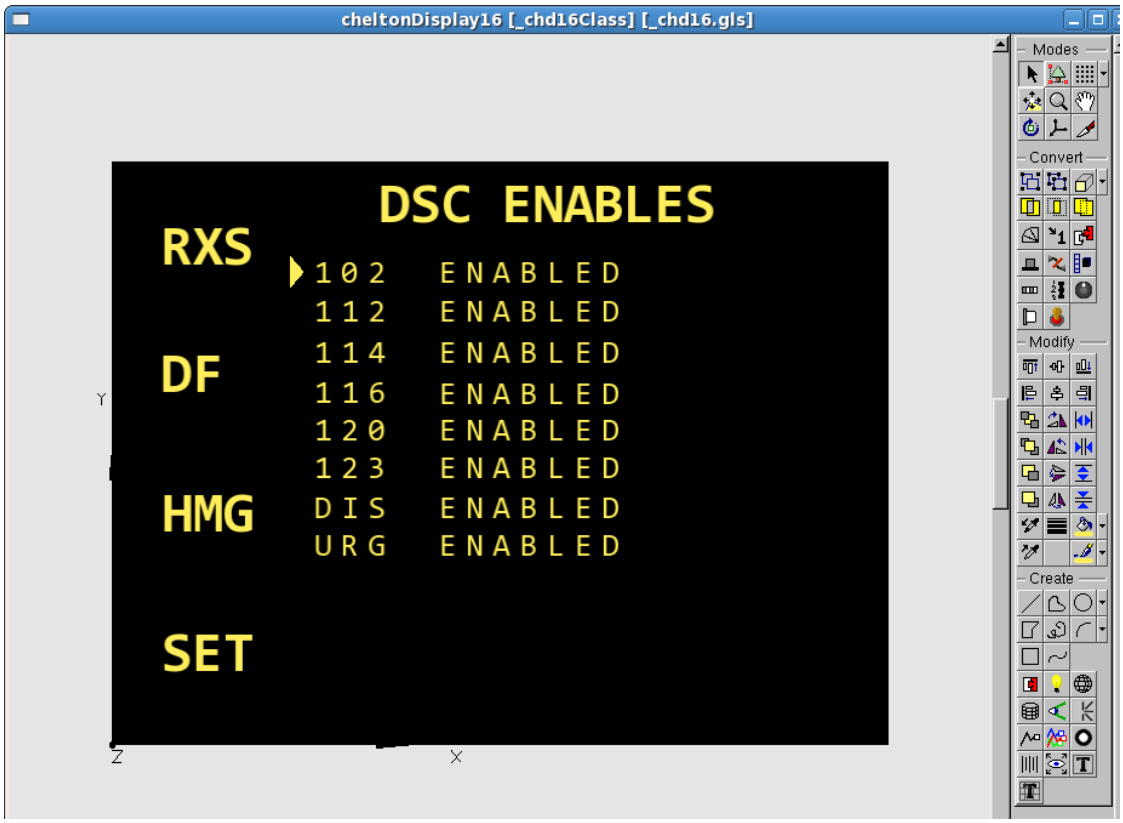


Figura 2.48: *Imagen de la pantalla 'DSC Enables Screen'*

Geometría utlizada

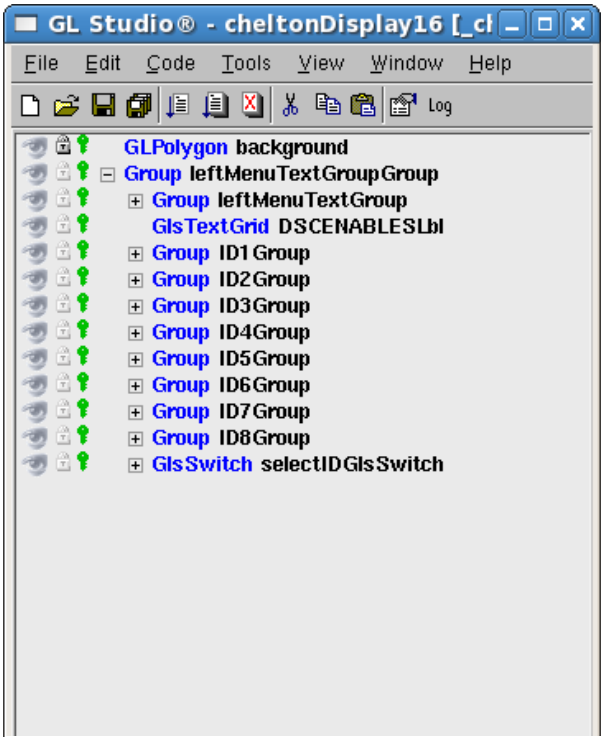


Figura 2.49: *Geometría de la pantalla 'DSC Enables Screen'*

La geometría utilizada para la pantalla 16 es la siguiente:

- Grupo *leftMenuTextGroup:* Etiquetas del menú lateral izquierdo.

- *ID1Group, ID2Group, ID3Group, ID4Group, ID5Group, ID6Group, ID7Group, ID8Group:* Grupos que contienen las etiquetas de la izquierda y su correspondiente estado (*DISABLES or ENABLED)*

Variables y métodos utilizados

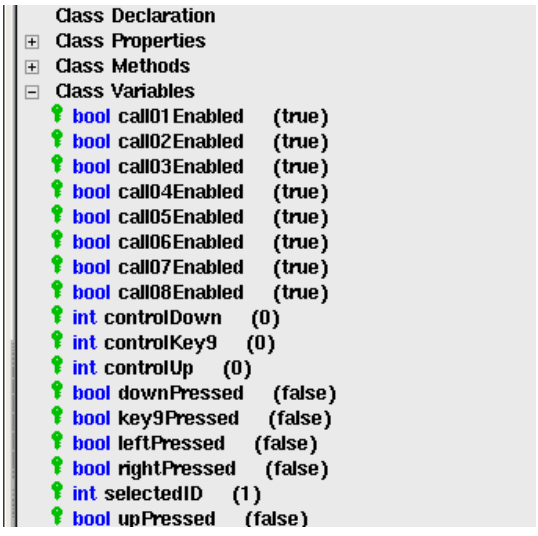


Figura 2.50: *Variables de la pantalla 'DSC Enables Screen'*

Las variables utilizadas son las siguientes:

- Grupo *leftMenuTextGroup:* Etiquetas del menú lateral izquierdo.

- *Call01Enabled, call02Enabled, call03Enabled, call04Enabled, call05Enabled, call06Enabled, call07Enabled, call08Enabled:* Variables que almacenan si están activos o inactivos los distintos campos.

- *controlDown, controlKey9, controlUp*: Almacenan si los botones se pulsan una única vez.

- *downPressed, key9Pressed, upPressed:* Indican cuándo se pulsa cada botón.

Métodos utilizados

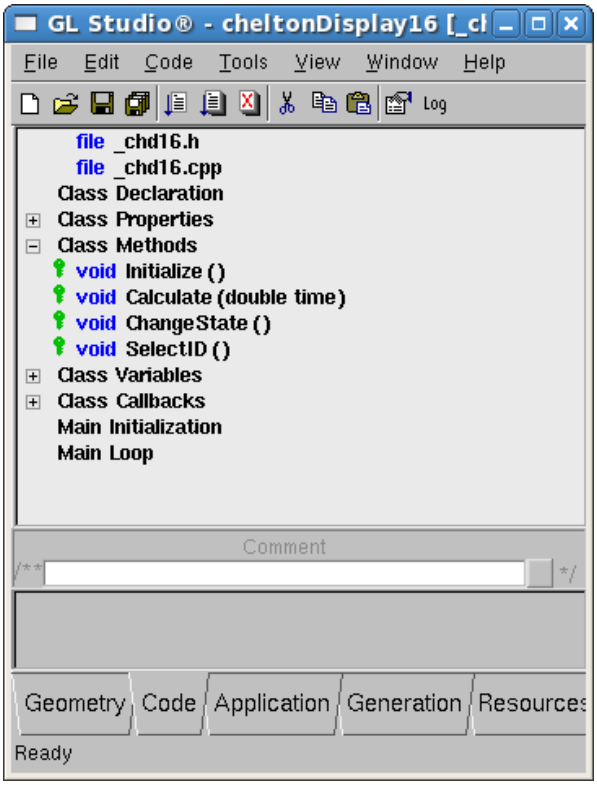


Figura 2.51: *Métodos de la pantalla 'DSC Enables Screen'*

La geometría utilizada para la pantalla 15 es la siguiente:

- *ChangeState():* Función que cambia los estados de *Enabled* a *Disabled* y viceversa, cuando el usuario pulsa *Enter*.

- *SelectID():* Función que recoge en qué lugar está el switch en ese momento.

Pantalla 17 – SU DF

Desde esta pantalla podemos cambiar parámetros como el volumen del receptor *DF*, seleccionar el receptor que va a tener la capacidad de reconocer múltiples balizas y otros parámetros.

El volumen puede ser incrementado o decrementando, en un rango de valores de 0 a 15, usando las teclas de *arriba* | *abajo* e *izquierda* | *derecha*, o el potenciómetro.

El *mounting attitude* puede ser cambiado presionando la tecla *MNT*.

El receptor que tiene la recepción de multi-balizas activo, puede ser cambiado presionando la tecla *MBR*, alternando entre *Manual, VHF, UHF* o *OFF*.

Los parámetros son sintonizados en el controlador.

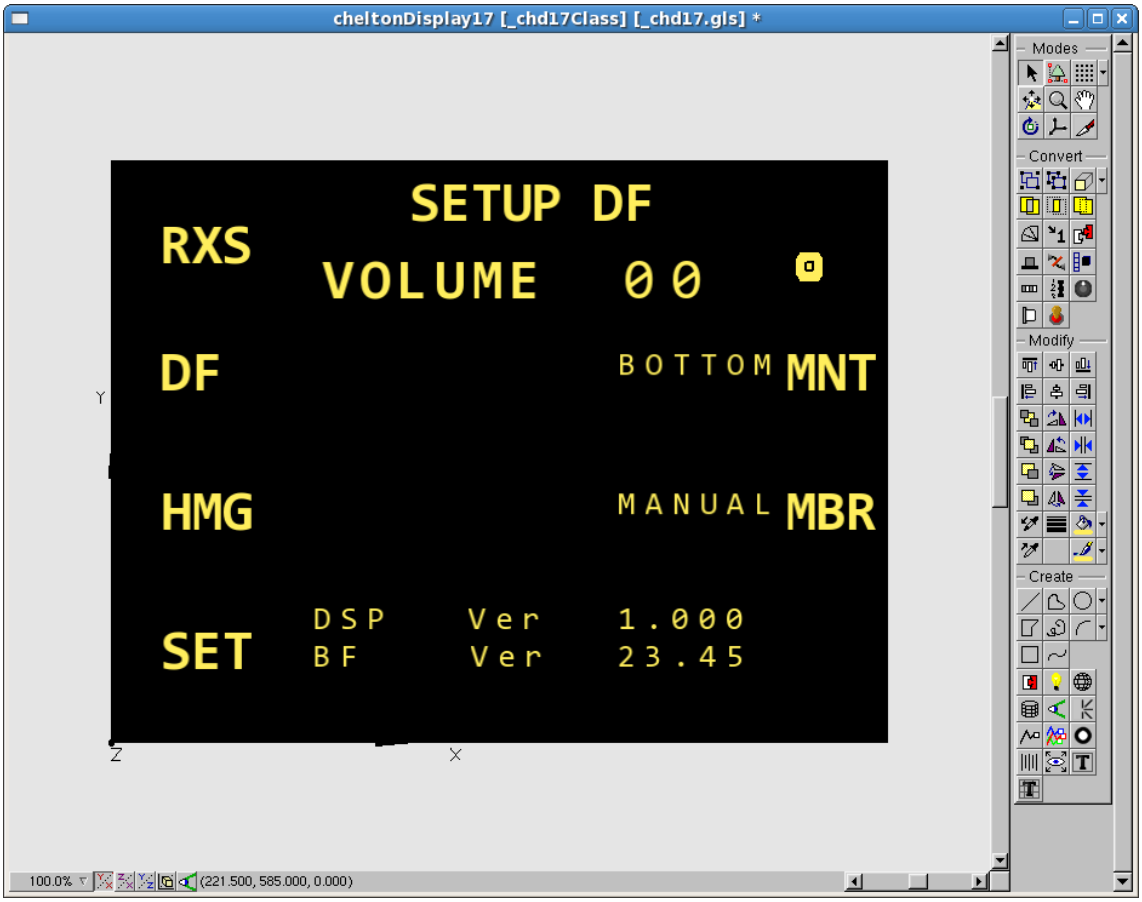


Figura 2.52: *Imagen de la pantalla 'SU DF Screen'*

Geometría utilizada

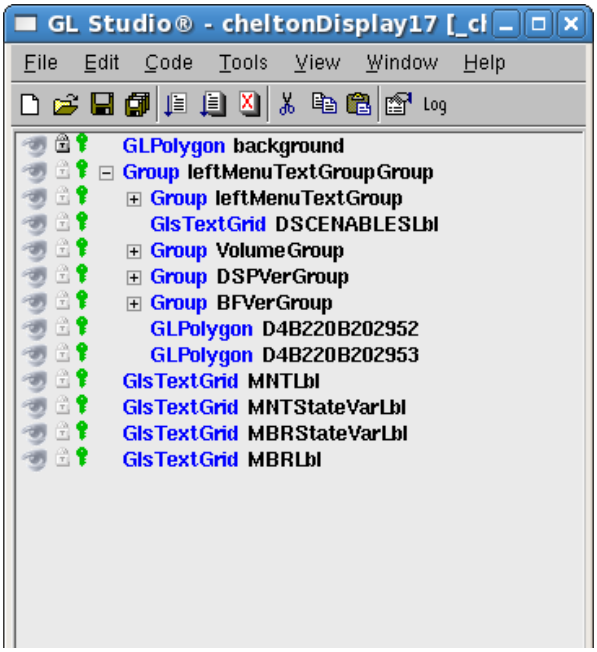


Figura 2.53: *Geometría de la pantalla 'SU DF Screen'*

La geometría utilizada para la pantalla 17 es la siguiente:

- Grupo *leftMenuTextGroup:* Etiquetas del menú lateral izquierdo.
- *VolumeGroup:* Grupo que contiene todo lo relacionado con el Volumen.
- *DSPVerGroup y BFVerGroup:* Contiene todas las etiquetas de las versiones de los mismos.
- *MNTLbl, MBRLbl*: Etiquetas de MBR y MNT.
- *MNTStateVarLbl, MBRStateVarLbl*: Valor de los campos MNT y MBR.

Variables utilizadas

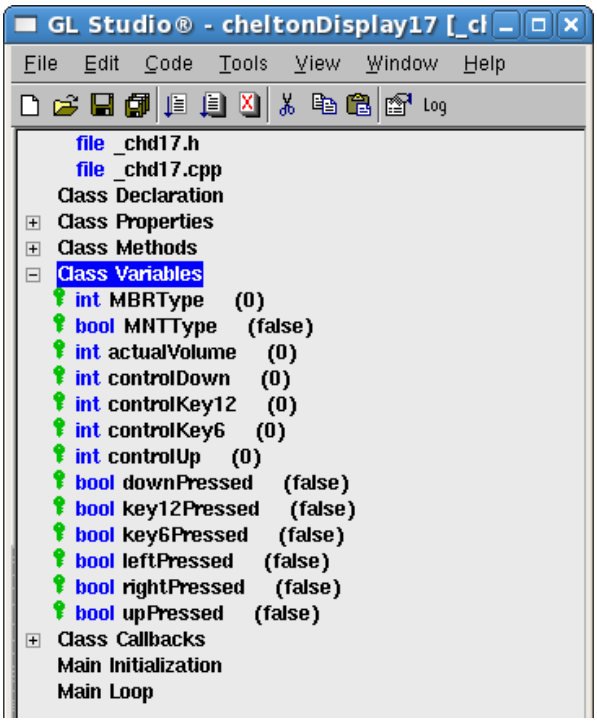


Figura 2.54: *Variables de la pantalla 'SU DF Screen'*

La geometría utilizada para la pantalla 17 es la siguiente:

- Grupo *leftMenuTextGroup:* Etiquetas del menú lateral izquierdo.
- *VolumeGroup:* Grupo que contiene todo lo relacionado con el Volumen.
- *DSPVerGroup y BFVerGroup:* Contiene todas las etiquetas de las versiones de los mismos.
- *MNTLbl, MBRLbl*: Etiquetas de MBR y MNT.
- *MNTStateVarLbl, MBRStateVarLbl*: Valor de los campos MNT y MBR.

Pantalla 18 – CSAR SETUP

Pantalla que muestra la pantalla de configuración del CSAR, pero que no permite en la actualidad cambiar ningún parámetro, debido a que no se contrató.

Figura 2.55: *Imagen de la pantalla 'SU CSAR Screen'*

Geometría utilizada

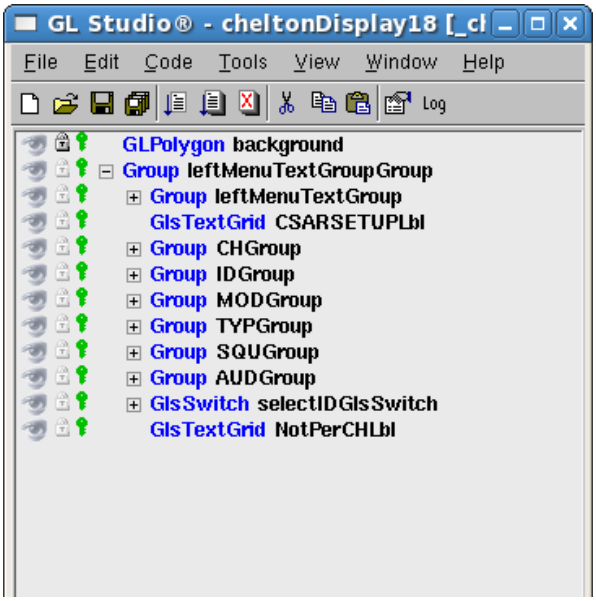


Figura 2.56: *Geometría de la pantalla 'SU CSAR Screen'*

La geometría utilizada para la pantalla 18 es la siguiente:

- Grupo *leftMenuTextGroup*: Contiene las etiquetas de los distintos menús de la izquierda.
- Grupos *CHGroup,IDGroup, ModGroup, TypGroup, SquGrooup, AUDGroup:* Contienen las distintas etiquetas de cada uno de los campos que tienen su nombre.
- *SelectIDGlsSwitch:* Switch que contiene el cursor con el que seleccionamos cada ítem.

Variables Utilizadas

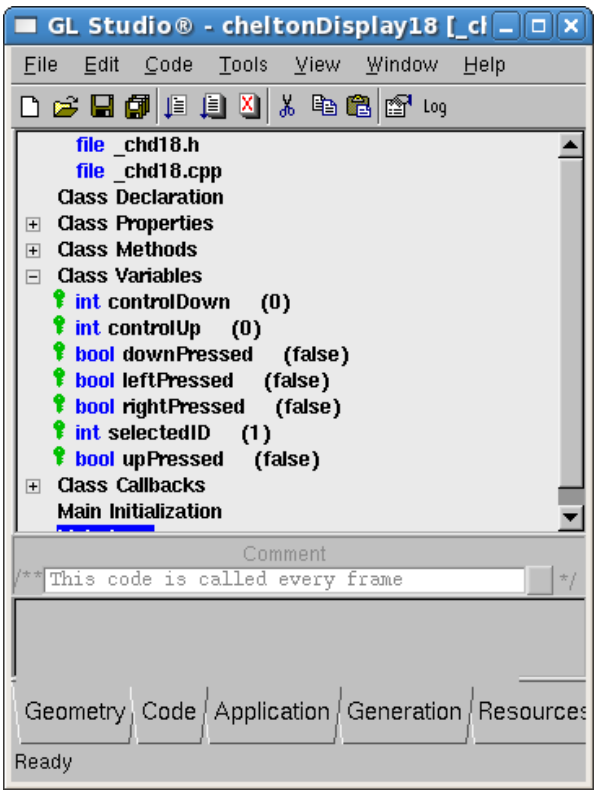


Figura 2.57: *Variables de la pantalla 'SU CSAR Screen'*

Las variables utilizadas son:

- *controlDown, conttrolUP:* Variables auxiliares que controlan la pulsación una sola vez del botón *down* y *up.*
- *downPressed,upPressed, leftPressed, rightPressed:* Variables que nos indican si se ha pulsado la tecla *abajo, arriba, izquierda* o *derecha*.
- *selectedID:* Variable que recoge la posición del cursor.

Métodos utilizados

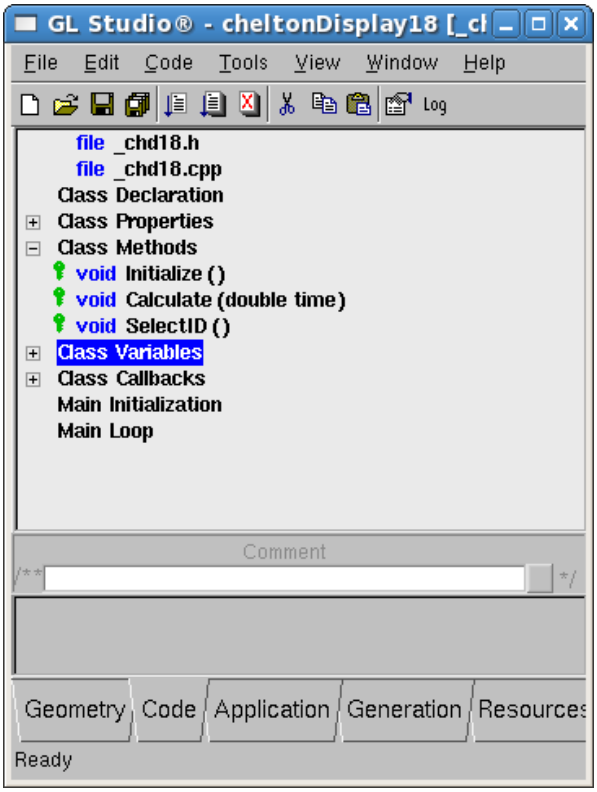


Figura 2.58: *Métodos de la pantalla 'SU CSAR Screen'*

Los métodos utilizados son:

- *SelectID:* Encargado de realizar las operaciones necesarias para movernos con el cursor por los distintos campos.

Pantalla 19 – SU CONT

El indicador de on-top para la pantalla de *DF* o *HOMING* puede ser cambiado entre *'flashing'* y *'continuous'* usando la tecla superior derecha.

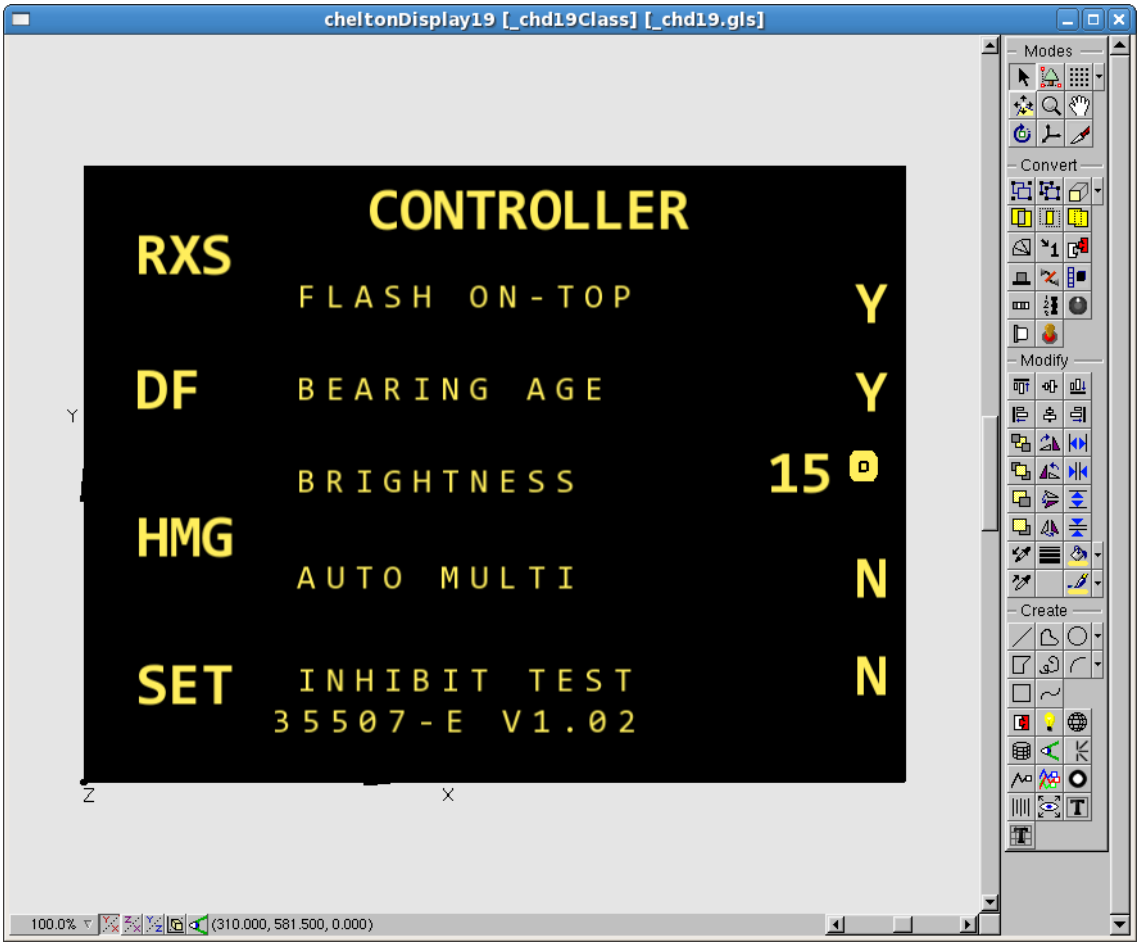


Figura 2.59: *Imagen de la pantalla 'CONTROLLER Screen'*

Geometría utilizada

La geometría utilizada para la pantalla 19 es la siguiente:

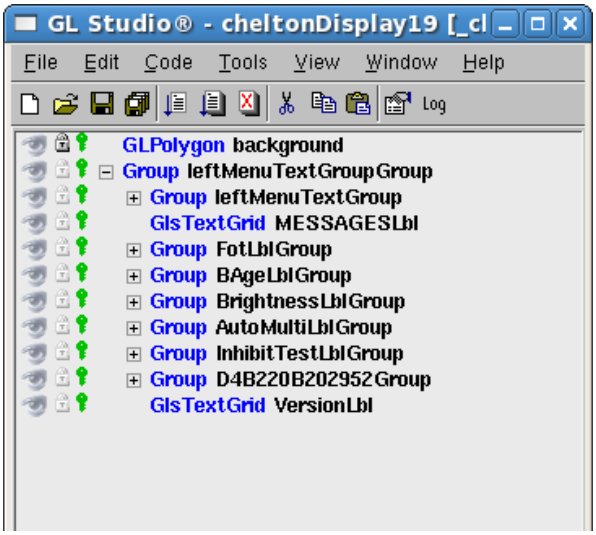


Figura 2.60: *Geometría de la pantalla 'CONTROLLER Screen'*

- Grupo *leftMenuTextGroup*: Contiene las etiquetas de los distintos menús de la izquierda.
- Grupos *FotLblGroup, BAgeLblGroup, BrightnessLblGroup, AutoMultiLblGroup, InhibitTextLblGroup:* Contienen las distintas etiquetas de cada uno de los campos que tienen su nombre.

Variables utilizadas

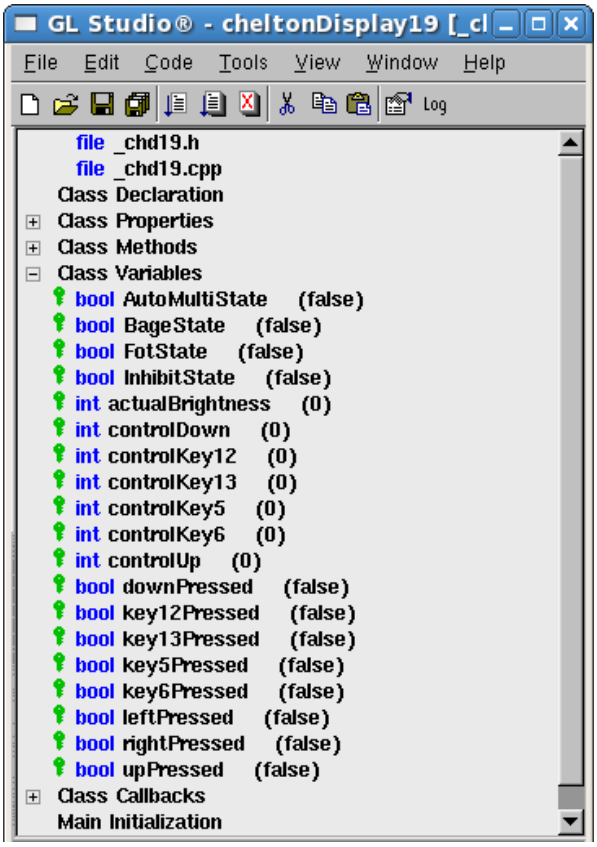


Figura 2.61: *Variables de la pantalla 'CONTROLLER Screen'*

Las variables utilizadas son:

- *controlDown, conttrolUP, controlKey5, controlKey6, controlKey12, controlKey13:* Variables auxiliares que controlan la pulsación una sola vez del botón *down* y *up.*
- *downPressed,upPressed, leftPressed, rightPressed, key5Pressed, key6Pressed, key12Pressed, key13Pressed:* Variables que nos indican si se ha pulsado la tecla *abajo, arriba, izquierda* o *derecha*.
- *selectedID:* Variable que recoge la posición del cursor.

Métodos utilizados

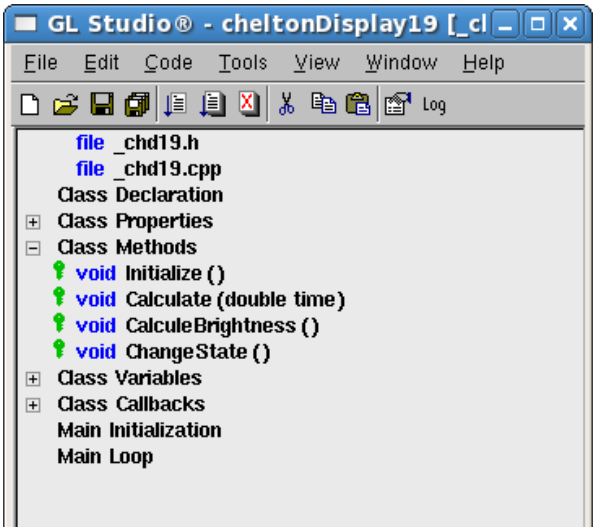


Figura 2.62: *Métodos de la pantalla 'CONTROLLER Screen'*

Los métodos utilizados son:

- *CalculateBrightness:* Método encargado de cambiar el valor del campo *Brightness.*
- *ChangeState*: Método que se encarga de cambiar el estado de los 4 distintos campos, de *Y* a *N* , y viceversa.

2.1.3. ESTRUCTURA DEL CHELTON DF 935

Según la metodología de trabajo que se sigue en la empresa INDRA, un departamento es el encargado de proporcionarme la disposición de los botones junto con el nombre de las variables, para que, en una fase posterior del proyecto, sin necesidad de comunicarnos entre los distintos departamentos, el aparato lograse funcionar correctamente, o con el mínimo número de errores.

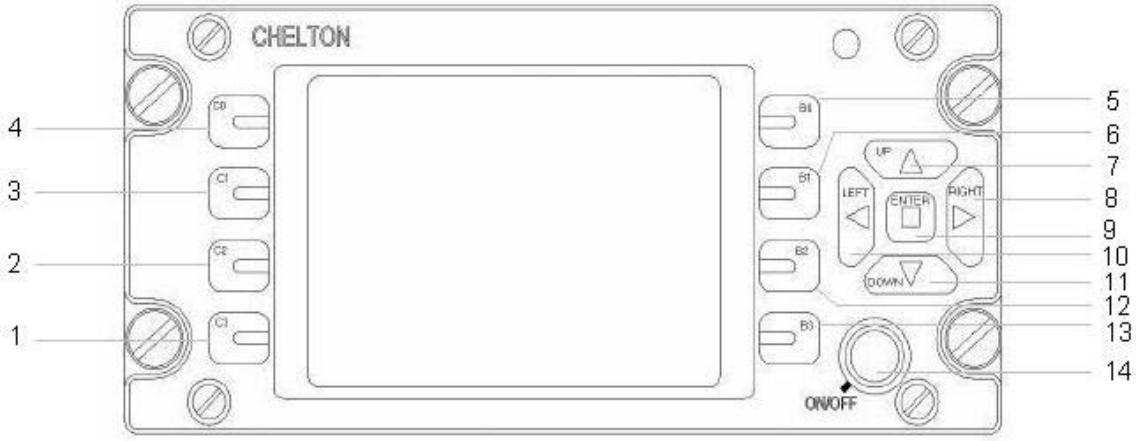


Figura 2.63: *EC225: CheltonDF935*

Struct name: st_chelton935DfIn

Description	Name	Type	Range	Default	I/O
Power Signal	PowerSignal	char	0: OFF 1: ON	0	I
(1) Key 1 pushbutton	pbKey1	char	0: NON PRESSED 1: PRESSED	0	I
(2) Key 2 pushbutton	pbKey2	char	0: NON PRESSED 1: PRESSED	0	I
(3) Key 3 pushbutton	pbKey3	char	0: NON PRESSED 1: PRESSED	0	I
(4) Key 4 pushbutton	pbKey4	char	0: NON PRESSED 1: PRESSED	0	I
(5) Key 5 pushbutton	pbKey5	char	0: NON PRESSED 1: PRESSED	0	I
(6) Key 6 pushbutton	pbKey6	char	0: NON PRESSED 1: PRESSED	0	I

(7) Key Up pushbutton	pbKeyUp	char	0: NON PRESSED 1: PRESSED	0	I
(8) Key Right pushbutton	pbKeyRight	char	0: NON PRESSED 1: PRESSED	0	I
(9) Key Enter pushbutton	pbKeyEnter	char	0: NON PRESSED 1: PRESSED	0	I
(10) Key Left pushbutton	pbKeyLeft	char	0: NON PRESSED 1: PRESSED	0	I
(11) Key Down pushbutton	pbKeyDown	char	0: NON PRESSED 1: PRESSED	0	I
(12) Key 12 pushbutton	pbKey12	char	0: NON PRESSED 1: PRESSED	0	I
(13) Key 13 pushbutton	pbKey13	char	0: NON PRESSED 1: PRESSED	0	I
(14) Chelton Power on pushbutton	pbCheltonPower	char	0: NON PRESSED 1: PRESSED	0	I

Tabla 2.1: *Estructura st_chelton935DfIn*

2.2. RESOLUCIÓN DE DUDAS TRAS PRIMERA LECTURA

Tras una primera lectura de la documentación, realizada entre el 10 de Octubre de 2009 hasta el 18 de Octubre de 2009, surgieron múltiples dudas, que solventamos con una ronda de preguntas a uno de los responsables del proyecto del helicóptero ***EC225*** en Indra, *Jorge Balbás Rodrigo*, que hizo una visita a León el día 18 de Octubre de 2009.

Ese día, tuvimos una presentación física por primera vez, ya que, habíamos contactado únicamente por correo, y me explicó exactamente cuáles eran los objetivos del simulador del *Chelton DF 935.*

El proyecto consiste en realizar una simulación, bajo linux, en C++, del funcionamiento del *Chelton DF 935*. Estaba dividido fundamentalmente en dos partes : la interfaz gráfica, junto con el botoneo, y la lógica del aparato, desde la cual se hallan los cálculos matemáticos.

Una vez recibida la información correcta sobre el trabajo a realizar, procedí a realizar las preguntas pertinentes surgidas durante la primera lectura. Fueron resueltas de forma rápida por *Jorge Balbás*. Las dudas eran las siguientes:

1. Desconocimiento de las acciones que debían realizar algunos botones, como por ejemplo, el botón *SLW* en las pantallas 3 y 4 (*DF* y *HMG*).
2. Dudas sobre la lógica del aparato (cálculos matemáticos a la hora de hallar el ángulo del rumbo del avión con respecto a la baliza, para mostrarlo correctamente en pantalla, entre otras).
3. Dudas sobre la información mostrada en los menús de "mensajes" tanto *DSC* como *COSPAS / SARSAT*.
4. Y otras pequeñas dudas sobre alguna acción a realizar en alguna pantalla.

Una vez realizada la primera toma de contacto con el proyecto, el siguiente paso fue reunirme con mi tutor de proyecto y comenzar a estudiar la metodología de trabajo para la realización del proyecto.

2.3. ELECCIÓN DE METODOLOGÍA DE TRABAJO PARA EL DESARROLLO

Una vez obtenida la información correcta y completa sobre el trabajo a realizar, me reuní con mi tutor de proyecto, *Miguel Ángel Benítez Andrades* y me explicó la metodología de trabajo a la hora de crear nuevos módulos del helicóptero, cómo poder encarar lo que proponían.

En primer lugar, realizamos un análisis sobre las posibles fases en las cuales se podía dividir el proyecto, de lo más general a lo particular, y se llegó a la siguiente conclusión:

- Realización de la interfaz gráfica (en adelante me referiré a ello como *Chelton*).
- Realización de la lógica del programa (de aquí en adelante, me referiré a esta parte como *módulo*).
- Comunicación entre el *Chelton* y el *módulo* mediante TCP.

Para ello, la parte gráfica se realizará con ayuda de la herramienta *glStudio* la cual explicaré más adelante, y el módulo se realizará en C++ con ayuda de un ide gráfico, en este caso, *NetBeans*.

2.4. ELECCIÓN DE FECHAS PARA CADA FASE

E l proyecto debía estar listo para realizar las primeras pruebas en el simulador real del helicóptero *EC225* en las instalaciones de *Indra San Fernando* para finales de Enero, principios de Febrero.

Para la realización del *Chelton*, la fecha conveniente para tener correctamente finalizadas las pantallas junto con el sistema de botoneo, era antes de las vacaciones de Navidad.

Teniendo así, un mes, para la realización del módulo del chelton, en el cual, surgirán dudas en los cálculos a realizar, y además, la creación de la comunicación del *Chelton* y el *módulo* por puertos TCP.

3. DESARROLLO DEL SIMULADOR

3.1. FASE 1: DESARROLLO DE INTERFAZ GRÁFICA CON GLSTUDIO

Para la realización del *Chelton*, se decidió utilizar la herramienta de trabajo *glStudio*. Esta herramienta, tiene se divide principalmente en tres zonas bien diferenciadas:

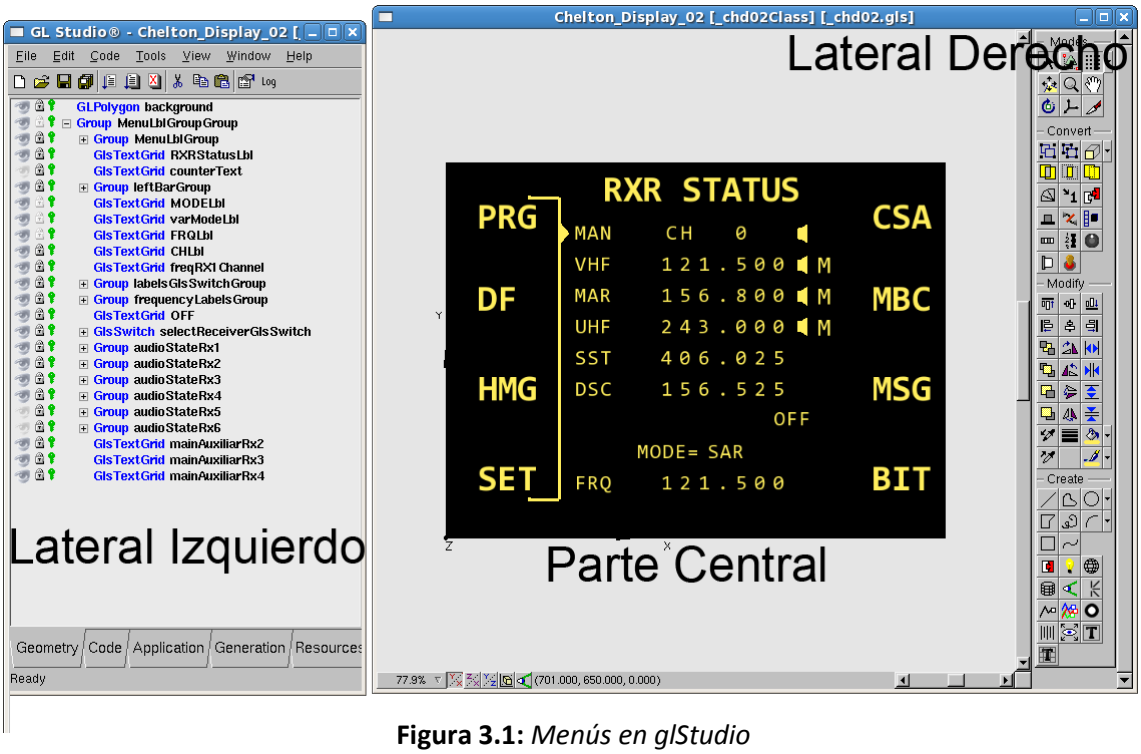


Figura 3.1: *Menús en glStudio*

- Menú lateral izquierdo: Contiene distintas pestañas por las cuales podemos navegar, en cada pestaña tenemos la siguiente información:
    - Geometry: En ella podemos visualizar los distintos elementos gráficos que posee nuestra pantalla.
    - Code: Desde esta, podemos cambiar todo el código que deba contener la pantalla que realizamos, crear, editar y borrar todo tipo de variables y métodos.
    - Application: Seleccionamos el tamaño, y otra serie de parámetros para generar la pantalla, como por ejemplo, que contenga bordes, que permita redimensionarla, etc.
    - Generation: Opciones para el código generado.
    - Resources: Más opciones.
- Zona central: En ella podemos encontrar la imagen o el menú con el cual trabajaremos para obtener el resultado que nosotros deseamos, uniendo los polígonos, botones y switches dibujados, con sus diferentes acciones.

- Menú lateral derecho: Consistente en una paleta de dibujo, con distintas herramientas, explicadas a continuación.

Esta herramienta es muy potente, es de pago, y la licencia está instalada en los ordenadores del CES en León. Con lo cual, para la realización de esta fase, era estrictamente necesario, realizar el trabajo desde las instalaciones de Indra.

3.1.1. CREACIÓN DE LAS PANTALLAS DE MENÚ

Las pantallas a realizar han sido 21, ya que, primero se creó la imagen del *Chelton* en su totalidad, es decir, pantalla + estructura con botones; seguidamente, se creó la pantalla que contenía las 19 pantallas (desde la cual se irán intercambiando cuando sea conveniente); y por último las 19 pantallas explicadas anteriormente, de forma individual y con sus acciones bien definidas.

Las pantallas realizadas han sido las siguientes:

Chelton_SES.gls

Contiene agrupadas todo el resto de pantallas, es el fichero que acaba generando el ejecutable.

cheltonDisplay.gls

Clase que se utiliza como nexo entre Chelton_SES y las 19 pantallas.

chd01.gls

En la primera pantalla, la única funcionalidad programada que existe, es la de, calcular el tiempo, y cambiar de pantalla cuando pasan 4 segundos.

chd02.gls

En la pantalla 2, anteriormente mostrada los métodos programadas en C++ son los siguientes:

SelectReceiver();

Método que contiene los algoritmos necesarios para poder cambiar de receptor seleccionado en esta pantalla. Se conseguirá pulsando los botones de arriba y abajo o con el potenciómetro.

SetFrequency();

Con este método, escribimos en la pantalla, la frecuencia que esté seleccionada de cada receptor, principal o auxiliar. El valor se recoge de la variable definida en el módulo, en el fichero Hmdfcheltonstructs_local.cpp

SetAudioState();

Este método dibuja el icono de un altavoz, en el caso de que en los receptores esté el audio activado, o no dibujara nada en caso contrario. Recoge el valor de la variable correspondiente.

SetMainAuxiliar();

Mediante este método, aparecerá en la pantalla una *M* o una *A* en cada receptor, dependiendo de si el usuario selecciona el auxiliar o el principal (main).

SignalDetected();

Este método subraya el título de cada receptor, cuando este detecta alguna baliza.

```
void _chd02Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

  SelectReceiver();
  SetFrequency();
  SetAudioState();
  SetMainAuxiliar();
  SetEditable();
  SignalDetected();

  if (1 == rxSelected) EditChannel();
  if (editable) EditMainAuxiliar();


}
```

Figura 3.2: *Código de chd02.gls*

chd03.gls

En la pantalla 3, los métodos programadas en C++ son los siguientes:

SelectReceiver();

Método que contiene los algoritmos necesarios para poder cambiar de receptor seleccionado en esta pantalla. Se conseguirá pulsando los botones de arriba y abajo o con el potenciómetro.

SetFrequency();

Con este método, escribimos en la pantalla, la frecuencia que esté seleccionada de cada receptor, principal o auxiliar. El valor se recoge de la variable definida en el módulo, en el fichero Hmdfcheltonstructs_local.cpp

SetAudioState();

Este método dibuja el icono de un altavoz, en el caso de que en los receptores esté el audio activado, o no dibujara nada en caso contrario. Recoge el valor de la variable correspondiente.

SetMainAuxiliar();

Mediante este método, aparecerá en la pantalla una *M* o una *A* en cada receptor, dependiendo de si el usuario selecciona el auxiliar o el principal (main).

SignalDetected();

Este método subraya el título de cada receptor, cuando este detecta alguna baliza.

```
void _chd03Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

  if ( key3Pressed ) {
    controlKey3++;

    if (controlKey3)
    {
      SlewFunction = !SlewFunction;
    }
  }
  else controlKey3 = 0;
```

```
if (( SlewFunction ) && ( CHELTON_OUT.receptorDetectedBeacons[rxSelected-1] > 0 ) )
{
   CHELTON_IN.mode = 1;
   SetManual();
}
else
{
   CHELTON_IN.mode = 0;
   SelectChannel();
   SelectReceiver();
   SetRxData();
   SetBeaconData();
   SignalDetected();
}
}
```

chd04.gls

En la pantalla 4, los métodos programadas en C++ son los siguientes:

SelectReceiver();

Método que contiene los algoritmos necesarios para poder cambiar de receptor seleccionado en esta pantalla. Se conseguirá pulsando los botones de arriba y abajo o con el potenciómetro.

SetFrequency();

Con este método, escribimos en la pantalla, la frecuencia que esté seleccionada de cada receptor, principal o auxiliar. El valor se recoge de la variable definida en el módulo, en el fichero Hmdfcheltonstructs_local.cpp

SetAudioState();

Este método dibuja el icono de un altavoz, en el caso de que en los receptores esté el audio activado, o no dibujara nada en caso contrario. Recoge el valor de la variable correspondiente.

SetMainAuxiliar();

Mediante este método, aparecerá en la pantalla una *M* o una *A* en cada receptor, dependiendo de si el usuario selecciona el auxiliar o el principal (main).

SignalDetected();

Este método subraya el título de cada receptor, cuando este detecta alguna baliza.

```
void _chd03Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

  if ( key3Pressed ) {
    controlKey3++;

    if (controlKey3)
    {
      SlewFunction = !SlewFunction;
    }
  }
  else controlKey3 = 0;


  if (( SlewFunction ) && ( CHELTON_OUT.receptorDetectedBeacons[rxSelected-1] > 0 ) )
  {
    CHELTON_IN.mode = 1;
    SetManual();
  }
  else
  {
    CHELTON_IN.mode = 0;
    SelectChannel();
    SelectReceiver();
    SetRxData();
    SetBeaconData();
    SignalDetected();
  }
}
```

chd05.gls

En la pantalla 5, los métodos programadas en C++ son los siguientes:

SelectBeacon();

Con este método, cambiamos el valor de la baliza seleccionada con los botones 5,6 y 12.

SetRxData();

Este método se encarga de insertar los distintos datos de cada receptor .

SetFirstBeaconData();

Método que calcula la diferente información de la baliza encontrada número uno. Realiza los cálculos de la distancia, la fuerza, la edad y el ángulo de la baliza respecto al helicóptero.

SetSecondBeaconData();

Método que calcula la diferente información de la baliza encontrada número dos. Realiza los cálculos de la distancia, la fuerza, la edad y el ángulo de la baliza respecto al helicóptero.

SetThirdBeaconData();

Método que calcula la diferente información de la baliza encontrada número tres. Realiza los cálculos de la distancia, la fuerza, la edad y el ángulo de la baliza respecto al helicóptero.

SetInfoBeacon();

Método que escribe en la pantalla, la información de la baliza seleccionada (1,2,3).

SignalDetected();

Este método subraya el título de cada receptor, cuando este detecta alguna baliza.

```
void _chd05Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

  SelectReceiver();
  SelectChannel();
  SelectBeacon();
  SetRxData();
  SetFirstBeaconData();
  SetSecondBeaconData();
  SetThirdBeaconData();
  SetInfoBeacon();
  SignalDetected();
}
```

chd06.gls

Esta pantalla, al ser una pantalla estática, carece de lógica, con lo cual, no posee ningún método.

```
void _chd06Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

}
```

chd07.gls

Esta pantalla, al ser una pantalla estática, carece de lógica, con lo cual, no posee ningún método.

```
void _chd07Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

}
```

chd08.gls

Esta pantalla, al ser una pantalla estática, carece de lógica, con lo cual, no posee ningún método.

```
void _chd08Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

}
```

chd09.gls

Esta pantalla contiene los siguientes métodos:

SelectCategory();

Esta función recoge el valor del switch correspondiente, y ese valor lo utiliza en el momento en el que el usuario pulsa el botón de *Rep*, yendo así, a unas pantallas u otras.

```
void _chd09Class::Calculate (double time)
{
   objects->Group::Calculate(time); // Do not remove (for normal operations)

   char aux[20];
   char aux2[20];
   SelectCategory();
}
```

chd10.gls

Esta pantalla, al ser una pantalla estática, carece de lógica, con lo cual, no posee ningún método.

```
void _chd10Class::Calculate (double time)
{
   objects->Group::Calculate(time); // Do not remove (for normal operations)

}
```

chd11.gls

Esta pantalla, al ser una pantalla estática, carece de lógica, con lo cual, no posee ningún método.

```
void _chd11Class::Calculate (double time)
{
   objects->Group::Calculate(time); // Do not remove (for normal operations)

}
```

chd12.gls

Esta pantalla, al ser una pantalla estática, carece de lógica, con lo cual, no posee ningún método.

```
void _chd12Class::Calculate (double time)
{
   objects->Group::Calculate(time); // Do not remove (for normal operations)

}
```

chd13.gls

Esta pantalla, al ser una pantalla estática, carece de lógica, con lo cual, no posee ningún método.

```
void _chd13Class::Calculate (double time)
{
   objects->Group::Calculate(time); // Do not remove (for normal operations)

}
```

chd14.gls

Esta pantalla contiene los siguientes métodos:

SetValues();

Este método es el encargado de asignar los distintos valores a las distintas características de cada receptor. Es decir, cuando activamos o desactivamos un receptor, asigna a la variable “encendido” un 1 o un 0 respectivamente. El mismo procedimiento sigue con el valor del Audio, y en el caso de las frecuencias, permite cambiar las frecuencias auxiliares de cada receptor en función del rango al que pertenezca.

SetChannel();

En el caso de que el receptor seleccionado sea el 1 (Manual), tenemos la posibilidad de cambiar el canal que queremos editar, esto se hace mediante este método.

CalculaRangos();

Esta función es utilizada para calcular el rango de valores de las frecuencias para cada uno de los receptores.

SetEditable();

Con este método, asignamos al a fariable *Editable* false o true, en función de las veces que el usuario pulse el botón *Enter*.

EditFieldsValues(int id);

Este método se encarga de editar, el elemento seleccionado, en función de la id del elemento que se le envíe como parámetro.

ChangeField();

Método mediante el cual nos movemos por los distintos campos para editar.

```
void _chd14Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

  SetValues();
  SetChannel();
  CalculaRangos();

  // Select Switch Control


  // Set Editable
  SetEditable();

  // Edit the fields
  if (editable) EditFieldsValues(selectedId);
  else ChangeField();

  //PrintValues();
}
```

chd15.gls

Esta pantalla, al ser una pantalla estática, carece de lógica, con lo cual, no posee ningún método.

```
void _chd15Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

}
```

chd16.gls

```
void _chd16Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

  SelectID();
  ChangeState();
}
```

chd17.gls

Esta pantalla contiene los siguientes métodos:

ChangeVolume();

Método encargado de variar el volumen entre 0 y 15, recogiendo la pulsación de las teclas *arriba* y *abajo* o mediante el potenciómetro.

ChangeMBR();

Método que cambia el valor del Receptor que detectará entre 1 y 3 balizas. Puede ser *Manual, VHF, UHF* o *OFF*.

ChangeMNT();

Método que cambia el valor del MNT entre *top* y *bottom*.

```
void _chd17Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

  ChangeVolume();
  ChangeMBR();
  ChangeMNT();
}
```

chd18.gls

```
void _chd18Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

  SelectID();
}
```

chd19.gls

Esta pantalla contiene los siguientes métodos:

ChangeState();

Mediante este método, dependiendo de las pulsaciones en los botones 5,6, 12 y 13, cambiará el valor de la cadena que tienen escrita a su altura enter *'Y' (yes)* y *'N' (no)*.

CalculeBrightness();

Método que varía el brillo entre 0 y 15, dependiendo del giro que realice el usuario al potenciómetro, o de las pulsaciones en las teclas *arriba* y *abajo*.

```
void _chd19Class::Calculate (double time)
{
  objects->Group::Calculate(time); // Do not remove (for normal operations)

  ChangeState();
  CalculeBrightness();
}
```

3.1.2. CREACIÓN DE FUNCIONES PARA EL FUNCIONAMIENTO DE LOS BOTONES EN CADA PANTALLA

Después de haber realizado el diseño y la programación de las 19 pantallas, el siguiente paso fue, comunicar a cada una de las pantallas, desde la principal, el valor de cada pulsación de botón, dependiendo de la pantalla en la que el usuario se encuentre.

Al ser una programación orientada a hilos de ejecución (constantemente se ejecutan todas las funciones que se han creado), el método más sencillo y fácil de implementar para dar la funcionalidad a cada botón en cada pantalla, fue el siguiente:

- Creación de un método en la pantalla principal, llamado *WhichMenu()*:

```
void Chelton_SESClass::WhichMenu ()
{
    /* WichMenu selects the menu when I press a key button that changes
     * the menu.Var controlScreen controls if mouse is left. When mouse
     * button is pressed, controlScreen = true, when mouse button is left
     * controlScreen = false
     */

    if (!controlScreen)
    {
    switch(actualScreen)
        {
          case 1:

          if(key2Pressed) {
              display->viewDisplay = 2;
              actualScreen = 2;
              controlScreen = true;
            }

            if(key3Pressed) {
              display->viewDisplay = 3;
              actualScreen = 3;
            controlScreen = true;
            }


          break;
          case 2:
            if (key1Pressed) {
              display->viewDisplay = 15;
              actualScreen = 15;
              controlScreen = true;
            }

            if(key2Pressed) {
              display->viewDisplay = 4;
              actualScreen = 4;
            controlScreen = true;
            }

            if(key3Pressed) {
              display->viewDisplay = 3;
              actualScreen = 3;
              controlScreen = true;
            }

            if((key4Pressed) && (display->cheltonDisplay02->rxSelected < 5)) {
              display->viewDisplay = 14;
              actualScreen = 14;
          controlScreen = true;
            }

            if(key5Pressed) {
              display->viewDisplay = 18;
              actualScreen = 18;
```

```
            controlScreen = true;
          }

          if(key6Pressed && display->cheltonDisplay17->MBRType < 3) {
            display->viewDisplay = 5;
            actualScreen = 5;
            controlScreen = true;
          }

          if(key12Pressed) {
            display->viewDisplay = 6;
            actualScreen = 6;
            controlScreen = true;
          }

          if(key13Pressed) {
            display->viewDisplay = 9;
            actualScreen = 9;
            controlScreen = true;
          }


        break;

        case 3:
          if (key1Pressed ) {
              actualScreen = 15;
              display->viewDisplay = actualScreen;
              controlScreen = true;
          }
          if (key2Pressed) {
              actualScreen = 4;
              display->viewDisplay = actualScreen;
              controlScreen = true;
          }
          if (key4Pressed) {
              actualScreen = 2;
              display->viewDisplay = actualScreen;
              controlScreen = true;
          }
        break;


        case 4:
          if (key1Pressed) {
              actualScreen = 15;
              display->viewDisplay = actualScreen;
              controlScreen = true;
          }
          if (key3Pressed) {
              actualScreen = 3;
              display->viewDisplay = actualScreen;
              controlScreen = true;
          }
          if (key4Pressed) {
              actualScreen = 2;
```

```
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
    break;


    case 5:
        if ( key1Pressed ) {
            actualScreen = 15;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key2Pressed ) {
            actualScreen = 4;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key3Pressed ) {
            actualScreen = 3;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key4Pressed ) {
            actualScreen = 2;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
    break;
    case 6:
        if ( key1Pressed ) {
            actualScreen = 15;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key2Pressed ) {
            actualScreen = 4;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key3Pressed ) {
            actualScreen = 3;
            display->viewDisplay = actualScreen;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key4Pressed ) {
            actualScreen = 2;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key5Pressed ) {
            actualScreen = 8;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key6Pressed ) {
```

```
            actualScreen = 7;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
    break;

    case 7:
        if ( key1Pressed ) {
            actualScreen = 6;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
    break;

    case 8:
    if ( key1Pressed ) {
            actualScreen = 6;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }

    break;

    case 9:
        if ( key1Pressed ) {
            actualScreen = 15;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key2Pressed ) {
            actualScreen = 4;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key3Pressed ) {
            actualScreen = 3;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key4Pressed ) {
            actualScreen = 2;
            display->viewDisplay = actualScreen;
            controlScreen = true;
        }
        if ( key6Pressed ) {
            actualScreen = display->cheltonDisplay09->categorySelected + 9;
            display->viewDisplay = actualScreen;
            controlScreen = true;

        }
    break;

    case 10:
    case 11:
    case 12:
    case 13:
```

```
if ( key1Pressed ) {
        actualScreen = 15;
        display->viewDisplay = actualScreen;
        controlScreen = true;
  }
  if ( key2Pressed ) {
        actualScreen = 4;
        display->viewDisplay = actualScreen;
        controlScreen = true;
  }
  if ( key3Pressed ) {
        actualScreen = 3;
        display->viewDisplay = actualScreen;
        controlScreen = true;
  }
  if ( key4Pressed ) {
        actualScreen = 2;
        display->viewDisplay = actualScreen;
        controlScreen = true;
  }
  if ( key5Pressed ) {
        actualScreen = 9;
        display->viewDisplay = actualScreen;
        controlScreen = true;
  }
break;


case 16:
case 17:
case 18:
case 19:
  if ( key1Pressed ) {
        actualScreen = 15;
        display->viewDisplay = actualScreen;
        controlScreen = true;
  }
  if ( key2Pressed ) {
        actualScreen = 4;
        display->viewDisplay = actualScreen;
        controlScreen = true;
  }
  if ( key3Pressed ) {
        actualScreen = 3;
        display->viewDisplay = actualScreen;
        controlScreen = true;
  }
  if ( key4Pressed ) {
        actualScreen = 2;
        display->viewDisplay = actualScreen;
        controlScreen = true;
  }

break;
```

```
case 14:
  if ( key1Pressed ) {
      actualScreen = 15;
      display->viewDisplay = actualScreen;
      controlScreen = true;
  }

  if ( key2Pressed ) {
      actualScreen = 4;
      display->viewDisplay = actualScreen;
      controlScreen = true;
  }

  if ( key3Pressed ) {
      actualScreen = 3;
      display->viewDisplay = actualScreen;
      controlScreen = true;
  }

  if ( key4Pressed ) {
      actualScreen = 2;
      display->viewDisplay = actualScreen;
      controlScreen = true;
  }

break;

case 15:
  if ( key2Pressed ) {
      actualScreen = 4;
      display->viewDisplay = actualScreen;
      controlScreen = true;
  }

  if ( key3Pressed ) {
      actualScreen = 3;
      display->viewDisplay = actualScreen;
      controlScreen = true;
  }

  if ( key4Pressed ) {
      actualScreen = 2;
      display->viewDisplay = actualScreen;
      controlScreen = true;
  }

  if ( key5Pressed ) {
      actualScreen = 16;
      display->viewDisplay = actualScreen;
      controlScreen = true;
  }

  if ( key6Pressed ) {
      actualScreen = 17;
      display->viewDisplay = actualScreen;
      controlScreen = true;
```

```
            }

            if ( key12Pressed ) {
                actualScreen = 18;
                display->viewDisplay = actualScreen;
                controlScreen = true;
            }

            if ( key13Pressed ) {
                actualScreen = 19;
                display->viewDisplay = actualScreen;
                controlScreen = true;
            }

          break;

    } // end_switch

  }
}
```

En este código se puede observar un *switch* ,el cual recoge el valor de la variable *screenActual* de modo que, dependiendo de la pantalla en la que el usuario se encuentre, y el botón que presione, cambiará a unas pantallas u a otras.

La relación de botoneo y pantallas es la siguiente:

- Pantalla 1:
    - Botón 1:
- Pantalla 2:
- Pantalla 3:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 4: Pantalla 2 – RXS Screen
- Pantalla 4:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen
- Pantalla 5:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen

- Pantalla 6:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen
    - Botón 5: Pantalla 7 – DSC Messages Screen
    - Botón 6: Pantalla 8 – SARSAT Messages Screen
- Pantalla 7:
    - Botón 1: Pantalla 6 – Message Screen
- Pantalla 8:
    - Botón 1: Pantalla 6 – Message Screen
- Pantalla 9:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen
    - Botón 6: Pantallas 10,11,12 y 13, dependiendo de un parámetro.
- Pantalla 10:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen
    - Botón 5: Pantalla 9 – Bite Screen
- Pantalla 11:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen
    - Botón 5: Pantalla 9 – Bite Screen
- Pantalla 12:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen

    - Botón 5: Pantalla 9 – Bite Screen
- Pantalla 13:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen
    - Botón 5: Pantalla 9 – Bite Screen
- Pantalla 14:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen
- Pantalla 15:
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen
    - Botón 5: Pantalla 16 – SU DSC Screen
    - Botón 6: Pantalla 17 – SU DF Screen
    - Botón 11: Pantalla 18 – CSAR Setup Screen
    - Botón 12: Pantalla 19 – Controller Screen
- Pantalla 16:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen
- Pantalla 17:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen
    - Botón 4: Pantalla 2 – RXS Screen
- Pantalla 18:
    - Botón 1: Pantalla 15 – Setup Screen
    - Botón 2: Pantalla 4 – HMG Screen
    - Botón 3: Pantalla 3 – DF Screen

- Botón 4: Pantalla 2 – RXS Screen

- Pantalla 19:
  - Botón 1: Pantalla 15 – Setup Screen
  - Botón 2: Pantalla 4 – HMG Screen
  - Botón 3: Pantalla 3 – DF Screen
  - Botón 4: Pantalla 2 – RXS Screen

➢ El siguiente paso fue crear 19 métodos, uno por cada pantalla, mediante el cual se relaciona el botoneo con distintas variables locales de cada pantalla:
Screens con las 19 funciones….

3.2. FASE 2: CREACIÓN DE MÓDULO CON LA LÓGICA DEL DF 935

Después de crear toda la parte gráfica con glStudio, el siguiente paso ha sido la creación del módulo en C++ que contiene la lógica del aparato. En INDRA, existe un protocolo de ficheros para la creación de los diferentes módulos del helicóptero, en este caso, del *EC225*, para una mayor facilidad de búsqueda de errores, en el caso de que los hubiera.

Para el desarrollo del módulo, fue necesario la creación de los siguientes ficheros:

- Hmcheltondf.h
  - Fichero que contiene el puntero para acceder al resto de módulos, los atributos y los métodos del módulo.
- Hmcheltondf_parameters.h
  - Fichero que contiene los parámetros del sistema Chelton DF 935.
- Hmcheltondf_struct.h
  - Fichero que contiene las estructuras de entrada y salida del sistema Chelton.
- Hmcheltondf_structlocal.h
  - Fichero que contiene estructuras de entrada y salida del sistema Chelton DF 935, que además, es compartido con la parte gráfica realizada en glStudio.
- HmcheltondfBase.h
  - Fichero que contiene las declaraciones de las funciones del fichero HmcheltondfBase.cpp.
- HmcheltondfSystem.h
  - Fichero que contiene las declaraciones de las funciones utilizadas en el HmcheltondfSystem.cpp
- Hmcheltondf.cpp

- Fichero que conecta el módulo del Chelton con el resto de módulos.

- HmcheltondfBase.cpp
  - Clase Base del Chelton DF 935.
- HmcheltondfSystem.cpp
  - Fichero que contiene la funcionalidad del sistema Chelton DF 935.

3.2.1. ANÁLISIS Y BÚSQUEDA DE ALGORITMOS PARA CÁLCULOS NECESARIOS

El sistema Chelton DF 935, necesita la realización de una serie de cálculos para poder mostrar la información de la localización de las diferentes balizas localizadas. Uno de los cálculos a realizar es, hallar la distancia geodésica con respecto a la Tierra.

¿Qué es la geodesia?

Geodesia es una ciencia interdisciplinaria que utiliza sensores remotos transportados en satélites espaciales y plataformas aéreas y mediciones terrestres para estudiar la forma y las dimensiones de la Tierra, de los planetas y sus satélites así como sus cambios; para determinar con precisión su posición y la velocidad de los puntos u objetos en la superficie u orbitando el planeta, en un sistema de referencia terrestre materializado, y la aplicación de este conocimiento a distintas aplicaciones científicas y técnicas, usando la matemática, la física, la astronomía y las ciencias de la computación.

¿Cómo se calcula la distancia geodésica entre dos puntos?

La fórmula matemática para calcular la distancia geodésica entre el helicóptero y cualquier baliza que detecte, es la siguiente:

*valor = sin(lat1) * sin(lat2) + cos(lat1) * cos(lat2) * cos(long1 – long2)*

distancia = arcos(valor)

*distancia respecto al eje Terrestre = distancia * 111.302*

Siendo *lat1 y long1* la latitud y la longitud de la posición del helicóptero y siendo *lat2 y long2* la longitud de la posición de la baliza detectada.

Otro de los cálculos importantes a realizar, es el de, el radio de cobertura de recepción que posee el helicóptero. Se calcula son la siguiente función matemática:

$$Cobertura = \sqrt[2]{alt + (2 * alt * Radio\ de\ la\ Tierra)}$$

Siendo *alt* la altitud a la que se encuentra el helicóptero.

3.2.2. CONSTANTES Y FUNCIONES DEL MÓDULO CHELTON DF 935

- En primer lugar, las constantes definidas son las siguientes:

```
namespace CheltondfConstants
{

  // INDEX OF RECEIVERS
  enum
  {
    RX1=0,
    RX2,
    RX3,
    RX4,
    RX5,
    RX6
  };

  enum
  {
    DISABLED,
    ENABLED
  };

  enum
  {
    NORMAL=0,
    SLW
  };

  const int MAX_NUM_BEACONS = 5;
  const int NUM_RECEIVERS = 6;
  const float RX2_FREQ = 121.500;
  const float RX3_FREQ = 156.800;
  const float RX4_FREQ = 243.000;
  const float RX5_FREQ = 406.025;

  const int EARTH_RADIUS = 6371;
  const double DEGTORAD = 0.01745329;
  const double RADTODEG = 57.29577951;

  const double MAX_DETECTION_DISTANCE = 160934.4;

  //ARINC VALIDITY
  enum
  {
    NOT_VALIDE,
    VALIDE
  };

  //ARINC ADFMD
  const int MD_ADF = 0;

  //ARINC W_ADFR
  const int FREQ_ENTIERE = 0;

  //ARINC BFO
  const int BFO_OFF = 0;

  //ARINC PADW1
```

```
    const int PADW1_FAUX = 0;

    //ARINC SDI_032_ADF
    const int SDI_ADF1 = 2;

    //ARINC SSM BCD VALUES
    enum
    {
        SSM_BCD_PLUS,
        SSM_BCD_NCD,
        SSM_BCD_TEST,
        SSM_BCD_MINUS
    };

    //ARINC SSM BNR VALUES
    enum
    {
        SSM_BNR_FW,
        SSM_BNR_NCD,
        SSM_BNR_TEST,
        SSM_BNR_NO
    };

    //ARINC SSM HOMING VALUES
    enum
    {
        SSM_HOMING_FW,
        SSM_HOMING_NCD,


        SSM_HOMING_NOT_USED,
        SSM_HOMING_NO
    };

}

#endif    /* _HMCHELTONDF_PARAMETERS_H */
```

- El siguiente paso es definir las estructuras necesarias.

```
namespace CHELTONDFStructs
{
  //============================================================================
  //                    :: CHELTON INPUTS ::
  //============================================================================

  // SELE IN
  struct st_InSele
  {
    bool bBkrDcPP25DF;
  };

  // SES IN
  struct st_InSES
  {
    float pmChelton;  // Range: 0 - 100%
    bool PowerSignal;       // 0 = OFF; 1 = ON
    bool pbKey1;           // 0 = NON PRESSED; 1 = PRESSED
    bool pbKey2;           // 0 = NON PRESSED; 1 = PRESSED
    bool pbKey3;           // 0 = NON PRESSED; 1 = PRESSED
    bool pbKey4;           // 0 = NON PRESSED; 1 = PRESSED
    bool pbKey5;           // 0 = NON PRESSED; 1 = PRESSED
    bool pbKey6;           // 0 = NON PRESSED; 1 = PRESSED
    bool pbKeyUp;           // 0 = NON PRESSED; 1 = PRESSED
    bool pbKeyRight;        // 0 = NON PRESSED; 1 = PRESSED
    bool pbKeyEnter;        // 0 = NON PRESSED; 1 = PRESSED
    bool pbKeyLeft;        // 0 = NON PRESSED; 1 = PRESSED
    bool pbKeyDown;         // 0 = NON PRESSED; 1 = PRESSED
    bool pbKey12;          // 0 = NON PRESSED; 1 = PRESSED
    bool pbKey13;          // 0 = NON PRESSED; 1 = PRESSED
    bool pbCheltonPower;    // 0 = NON PRESSED; 1 = PRESSED
  };

  // HEL IN
  struct st_InHel
  {
    double dLongitude;             //pEmov->longitud
    double dLatitude;             //pEmov->latitud
    double dAltitude;             //pEmov->Altitud
    double dBearingHel;             //pEmov->geographicDirection
  };
```

```
    /*
    // IOS IN
    struct st_InIOS
    {
        bool bADFFailure;
        bool bADFFault;
    };
    */

    // SAR IN
    struct st_InBeaconSAR
    {
        int index;              // Index of beacon
    bool state;             // State of beacon [0: DISABLED, 1: ENABLED]
    double bearingGeo;        // Geographic Bearing
    double bearingMag;        // Magnetic Bearing
    double altitude;          // Altitude of the beacon
    double latitude;          // Latitude of the beacon
    double longitude;         // Longitude of the beacon
    double speed;             // Speed of the beacon
    long frequency;           // Frequency of the beacon
    };

    struct st_cheltondf_in
    {
        st_InSele Sele;
        st_InSES Ses;
        st_InHel Hel;
        //st_InIOS Ios;
    };

    //============================================================================
    //                  :: CHELTON OUTPUTS ::
    //============================================================================
    struct st_OutSES
    {
        bool ltDisplayOn;      // Set display status [0: OFF, 1: ON]
        bool ltLed;          // Set led status [0: OFF, 1: ON]
    };

    struct st_cheltondf_out
    {


    //TO MFDs
        ADF_Arinc_eng_t MFDs;

        //TO RTIOS
        st_OutSES Ses;
    };
}
#endif    /* _HMCHELTON_STRUCT_H */
```

- Estructuras compartidas con la interfaz creada con *glStudio*.

```
//=========================================================================
//                  :: CHELTON DF STRUCTS ::
//=========================================================================

//=========================================================================
// SARSAT MESSAGE
//=========================================================================
struct st_msgSTT
{
  bool msgState;          // SST messages indications [0: UNREAD (B), 1: READ (^)]
                          // ---> Crearemos un método que comprobará los mensajes
                          //      que están sin leer...

  char sarsatData[16];      // Raw SARSAT data, 16 numeric characters
  bool msgNature;          // Nature of the message [0: TEST, 1:DISTRESS]
  char coutryInfo[7];       // Country information
  int auxiliaryLoc;        // Auxiliary location device <jabenitez>
  char protocol[3];         // Protocol used
  double longMsgBeacon;     // Longitud of the beacon in the message
  double latMsgBeacon;      // Latitude of the beacon in the message
};


//=========================================================================
// DSC MESSAGE
//=========================================================================
struct st_msgDSC
{
 // DSC Message
  bool msgState;          // DSC messages indications [0: UNREAD (B), 1: READ (^)]
                          // ---> Crearemos un método que comprobará los mensajes
                          //      que están sin leer...

  char FOR[3];            // The format of the message
  char CAT[3];            // The category identifier [100:ROUTINE, 110:URGENCY, 112: DISTREES]
  char CID[10];           // Called identifier (Numerical number)
  char SID[10];           // Self identifier (Numerical number)
  char DID[10];           // Distressed identifier (Numberical number)
  char NOD[3];            // Nature of distreessed (Number)
  char SUB[3];            // Subsequent Communications type.
  char TC1[3];            // Tele-Command 1
  char TC2[3];            // Tele-Command 2
  double LAT;             // Longitud sent by the message sender
  double LON;             // Latitude sent by the message sender
  float TIM;              // Time attached to the message
  char CSR[6];            // Called station, received channel or frequency
  char CST[6];            // Called station, transmit channel or frequency
};

//=========================================================================
// RX DATA
//=========================================================================

struct st_rxData
{
  long freqManual[10];     // RX1 - Manual receiver stores 10 channels [30-470Mhz]
```

```
  long freqGuard[5];       // RX2 - VHF [MAIN: 121.500Mhz , AUX: 120.000-130.000Mhz]
                           // RX3 - MAR [MAIN: 156.800Mhz , AUX: 150.000-160.000Mhz]
                           // RX4 - UHF [MAIN: 243.000Mhz , AUX: 240.000-250.000Mhz]
                           // RX5 - COSPAT/SARSAT [406.025Mhz]
                           // RX6 - Maritime Ch70 [156.525Mhz]

  bool audioState[6];      // RX Audio State [0: NORMAL, 1: MUTE]
  bool activated[6];       // RX activated [0: NON ACTIVATED, 1: ACTIVATED]
  bool mainAuxiliar[6];    // RX MAIN or AUXILIAR frequency [ 0: MAIN, 1: AUXILIAR]

  int type[6];           // PROG RX SCREEN - RX type [0:MANUAL, 1:VHF, 2:MAR, 3:UFH, 5:SARSAT, 6:CH70
  int sqType[6];         // PROG RX SCREEN - SQ.TYPE
  int level[6];          // PROG RX SCREEN - Level
  int BW[6];             // PROG RX SCREEN - B/W
  int MOD[6];            // PROG RX SCREEN - MOD

  int channelSelected;     // Channel Selected [0-9]

  int rxSelected;          // Receiver Selected [0-5]

};

//================================================================================
// BITE SCREEN
//================================================================================
struct st_screenBITE
{
  bool GEN;              // General [0: BAD, 1: GOOD]
  bool DF;               // Direction Finder [0: BAD, 1: GOOD]
  bool CONT;             // Controller [0: BAD, 1: GOOD]
  bool PSRI;             // PSRI [0: BAD, 1: GOOD]
};

//================================================================================
// GENERAL REPORT SCREEN
//================================================================================
struct st_screenGEN
{
  bool DF_RESP;          // DF Response [0: FAIL, 1: OK]
  bool DF_TO;            // DF [0: FAIL, 1: OK]
  bool D_TO_A;           // D to A [0: FAIL, 1: OK]
  bool EEPROM;           // EEPROM [0: FAIL, 1: OK]
  bool A_TO_D;           // A to D [0: FAIL, 1: OK]
};

//================================================================================
// DF REPORT SCREEN
//================================================================================
struct st_screenREP
{
  bool SYNTH;            // SYNTH [0: FAIL, 1: OK]
  bool PSU;              // PSU [0: FAIL, 1: OK]
  bool I2C;              // I2C [0: FAIL, 1: OK]
  bool AD;               // AD [0: FAIL, 1: OK]
  bool TEMP;             // TEMP [0: FAIL, 1: OK]
  bool ANTEN;            // ATEN [0: FAIL, 1: OK]
```

```
  bool COMMS;              // COMMS [0: FAIL, 1: OK]
};

//=============================================================================
// CONTROLLER REPORT SCREEN
//=============================================================================
struct st_screenCONT
{
  char CHSUM[5];           // CHSUM
  bool V5_12;             // 5/12 V [0: FAIL, 1: OK]
  bool EPROM;              // EPROM [0: FAIL, 1: OK]
  float TEMP;             // Temperature
};

//=============================================================================
// PSR REPORT SCREEN
//=============================================================================
struct st_screenPSR
{
  int PSR_FAULTS;          // PSR faults
};

//=============================================================================
// SU DF SCREEN
//=============================================================================
struct st_screenSUDF
{
  int dfAudioVolume;       // DF Audio Volume [0-15]
  bool mntPosition;        // The mounting attitude [0:TOP, 1:BOTTOM]
  int mbrType;            // The multi-beacon active [0:OFF, 1:MANUAL, 2:VHF, 3:UHF]
  bool multiBeaconState;    // The multi-beacon state [0:DISABLED, 1:ENABLED]
};

//=============================================================================
// SU DSC
//=============================================================================
struct st_screenSUDSC
{
  bool DSC_102;            // 102: Selective Call to a specific Geographic Area [0:DISABLED, 1:ENABLED
  bool DSC_112;            // 112: Distress Alert Call (ENABLED!) [0:DISABLED, 1:ENABLED]
  bool DSC_114;            // 114: Selective Call to a Common Interest Group [0:DISABLED, 1:ENABLED]
  bool DSC_116;            // 116: All Ships Call [0:DISABLED, 1:ENABLED]
  bool DSC_120;            // 120: Selective Call to an Individual Station [0:DISABLED, 1:ENABLED]
  bool DSC_123;            // 123: Automatic Selective Call to an Individual Stations [0:DISABLED, 1:EN
  bool DSC_DIS;            // DIS: Decode of all DISTRESS type messages (ENABLED!) [0:DISABLED, 1:ENA
  bool DSC_URG;            // URG: Decode of all URGENCY type messages [0:DISABLED, 1:ENABLED]
};

//=============================================================================
// SU CONTROLLER
//=============================================================================
struct st_screenSUCONT
{
  bool contFlashOnTop;      // Flash ON-TOP [0:NO, 1:YES]
  bool contBearingAge;      // Bearing Age  [0:NO, 1:YES]
  int  contBrightness;      // Brightness [0-15]
```

```
  bool contAutoMulti;       // Auto Multi Beacon [0:NO, 1:YES]
  bool contInhiTest;        // Inhibit Test [0:NO, 1:YES]
};

//=============================================================================
// SU CSAR
//=============================================================================
struct st_screenSUCSAR
{
  char channelId[6];          // ID of the channel
  char channelMod[6];         // MOD of the channel
  char channelTyp[10];        // Type of the channel
  char channelSqv[6];         // SQV of the channel
  char channelAud[6];         // AUD of the channel
};

//=============================================================================
// BEACON DATA
//=============================================================================

struct st_beacon
{
  float age;          // Age of the signal
  float strength;     // The signal strength of the beacon   [-140,-70dBm]

  float bearingRel;   // The relative bearing of the beacon (Helicopter)
  float bearingAbs;   // The absolute bearing of the beacon (NORTH)
  float distance;     // Distance to Helicopter

  bool state;       // State of the beacon [0: NON ACTIVE, 1: ACTIVE]
};

//=============================================================================
// BEACON DATA
//=============================================================================

struct st_OutSESCheltondf
{
  float pmChelton;    // Range: 0 - 100%
  bool PowerSignal;     // 0 = OFF; 1 = ON
  bool pbKey1;          // 0 = NON PRESSED; 1 = PRESSED
  bool pbKey2;          // 0 = NON PRESSED; 1 = PRESSED
  bool pbKey3;          // 0 = NON PRESSED; 1 = PRESSED
  bool pbKey4;          // 0 = NON PRESSED; 1 = PRESSED
  bool pbKey5;          // 0 = NON PRESSED; 1 = PRESSED
  bool pbKey6;          // 0 = NON PRESSED; 1 = PRESSED
  bool pbKeyUp;         // 0 = NON PRESSED; 1 = PRESSED
  bool pbKeyRight;      // 0 = NON PRESSED; 1 = PRESSED
  bool pbKeyEnter;      // 0 = NON PRESSED; 1 = PRESSED
  bool pbKeyLeft;       // 0 = NON PRESSED; 1 = PRESSED
  bool pbKeyDown;       // 0 = NON PRESSED; 1 = PRESSED
  bool pbKey12;         // 0 = NON PRESSED; 1 = PRESSED
  bool pbKey13;         // 0 = NON PRESSED; 1 = PRESSED
  bool pbCheltonPower;  // 0 = NON PRESSED; 1 = PRESSED
};

//=============================================================================
```

```
//                  :: END CHELTON DF STRUCTS ::
//=============================================================================

//=============================================================================
//                  :: CHELTON INPUTS ::
//=============================================================================

struct st_cheltonToHost
{
  st_rxData rxData;          // Receivers


  st_screenBITE screenBITE;     // BITE Screen Data
  st_screenGEN screenGEN;       // GEN Screen Data
  st_screenREP screenREP;       // REPORT Screen Data
  st_screenCONT screenCONT;     // CONTROLLER Screen Data
  st_screenPSR screenPSR;       // PSR Screen Data
  st_screenSUDF screenSUDF;     // SU DF Screen Data
  st_screenSUDSC screenSUDSC;   // SU DSC Screen Data
  st_screenSUCONT screenSUCONT; // SU CONT Screen Data
  st_screenSUCSAR screenSUCSAR; // SU CSAR Screen Data

  int actualScreen;          // Screen is displayed

  int mode;                  // Modes of Chelton [0 = NORMAL, 1 = SLW]

  float bearingRelManual[4]; // Bearing Manual for SLEW MODE
};
//=============================================================================
//                  :: CHELTON OUTPUTS ::
//=============================================================================

struct st_hostToChelton
{
  st_OutSESCheltondf Ses;

  st_msgSTT cheltonSTT[8];   // STT Chelton Messages
  st_msgDSC cheltonDSC[8];   // DSC Chelton Messages

  st_msgSTT dfSTT[9];        // STT DF Messages
  st_msgDSC dfDSC[9];        // DSC DF Messages
                  // HOST-DF to CHELTON

  st_beacon beacon[5];       // 4 Boats & 1 Cast Away
                  // 5 Elements & 4 RX =

  int unreadCounterSTT;      // Number of unreaded SST messages
  int totalCounterSTT;       // Total SST messages

  int unreadCounterDSC;      // Number of unreaded DSC messages
  int totalCounterDSC;       // Total DSC messages

  bool beaconDetected[4][5]; // Flag to indicate if a beacon has been detected
        // to MAX_NUM_BEACONS

  int indexBeaconsDetected[4][3]; // Indexes of beacons with more strength detected
```

```
    int numBeaconsDetected[4];  // Save the number of beacons detected by a receptor
    bool receptorDetectedBeacons[4]; // Save if a receptor has detected a beacon
  };

  struct st_chelton_EC225
  {
    st_hostToChelton hostToChelton;
    st_cheltonToHost cheltonToHost;
  };
}
#endif    /* _HMCHELTON_STRUCTLOCAL_H */
```

- Las funciones principales del módulo principal son las dos siguientes:

```
////////////////////////////////////////////////////////////////////////////////
//*****************************************************************************//
//
// FUNCTION:  ModuleExecution()
//
// DESCRIPTION: Execution of the module.
//
// ARGUMENTS: ---
//
// RETURN: ---
//
//*****************************************************************************//
////////////////////////////////////////////////////////////////////////////////

void cl_CheltondfSystem::ModuleExecution(void)
{
  int channelSelected = 0;
  float frequency = 0;

  cheltonWorks = Cheltondf_OK();

  //Check if system can work
  if ( cheltonWorks )
  {

    FillSesOutputs();

    // Set data about altitude, longitude and latitude of the HEL
    m_Receptor.SetElevacion(input->Hel.dAltitude);
    m_Receptor.SetLatLon(input->Hel.dLatitude, input->Hel.dLongitude);

    // Get radius of cover
    coverRadius = GetCover();

    // FOR EACH OF RECEIVERS do...
    for (int receiver = 0; receiver < NUM_RECEIVERS-2; receiver++)
    {
      // Init nº of beacons detected
      InitNumBeaconsDetected(receiver);

      // If the receiver is ACTIVATE
      if (chelton_in->rxData.activated[receiver])
```

```
        {
          // Get the channel selected
          channelSelected = chelton_in->rxData.channelSelected;

          // Get the frequency selected
          frequency = CalculateFrequencySelected(receiver, channelSelected);

          // Caculate the beacon data for this receiver
          CalculateBeaconData(receiver, frequency);

          // Set the beacons detected
          SetBeaconsRxDetected(receiver, frequency);
        }
        else NoneBeaconsDetected(receiver);

        // Get nº of beacons detected
        GetNumBeaconsDetected(receiver);
      } // for (int receiver = 0; receiver < NUM_RECEIVERS-2; receiver++)
    } // End if cheltonOk

    FillOutputs();

} // End of ModuleExecution()


////////////////////////////////////////////////////////////////////////////
//**************************************************************************//
//
// FUNCTION:  CalculateBeaconData
//
// DESCRIPTION: Calculate all data of 5 beacons compare to receivers
//
// ARGUMENTS: --
//
// RETURN: --
//
//**************************************************************************//
////////////////////////////////////////////////////////////////////////////

void cl_CheltondfSystem::CalculateBeaconData(int rx, float frequencySelected)
{


    // For 0 to 4 ( We receive data of 5 beacon [4 boats & 1 Cast away])
    for (int index = 0; index < MAX_NUM_BEACONS; index++)
    {
      chelton_out->beacon[index].state = input->beaconSar[index].state;

      // Beacon not detected (by default)!!
      chelton_out->beaconDetected[rx][index] = false;

      // Beacon ACTIVE
      if (chelton_out->beacon[index].state)
      {
        // Set data about a beacon
        m_PositionBeacon.SetLatLon(input->beaconSar[index].latitude, input->beaconSar[index].longitude);
```

```
        // Compare the beacon frequency with frequency selected
      if (input->beaconSar[index].frequency == frequencySelected)
        {

          // Calcule the geodesic distance relative to Hel
          chelton_out->beacon[index].distance = GetGeodesicDistance(input->Hel.dLatitude,
                                                input->Hel.dLongitude,
                                                input->beaconSar[index].latitude,
                                                input->beaconSar[index].longitude);

          // If the geodesic distance is < coverRadius ...
          if( chelton_out->beacon[index].distance < coverRadius)
          {
            // Beacon detected!!
            chelton_out->beaconDetected[rx][index] = true;

            // Calcule the bearing relative, age and the strength (inverse to distance)
            chelton_out->beacon[index].bearingRel = m_Receptor.RadialRelativo(input->beaconSar[ind
            chelton_out->beacon[index].bearingRel = fmod(chelton_out->beacon[index].bearingRel - inp
                                  + 360 + 180, 360) - 180;
            chelton_out->beacon[index].age = 0;
            chelton_out->receptorDetectedBeacons[rx] = true;
            chelton_out->beacon[index].strength = 1/chelton_out->beacon[index].distance;
          } // end if beacon.distance < coverRadius


        } // end_if (input->beaconSar[index].frequency == frequencySelected)
      // Sort the beacons by strength
      SortBeacons();

    } // End_for (int index = 0; index < MAX_NUM_BEACONS; index++)

}
```

3.2.3. CÓDIGO DE GLSTUDIO PRINCIPAL

Desde la pantalla principal del Chelton realizada en *glStudio*, se debe enviar una serie de información sobre los botones que son pulsados en cada momento, esto se consigue mediante una serie de funciones creadas específicamentes para cada pantalla.

En la variable *screenActual* se almacena la pantalla en la que se encuentra el usuario en cada momento. Con ella, en la función principal de *Chelton_SES.gls* mediante una serie de *ifs*, seleccionamos qué acciones queremos que estén ejecutándose, en función de la pantalla. Las funciones están nombradas de la forma *screenXYActions();*

Además de enviar información de los botones pulsados, posee también un control de variables necesarias en más de una pantalla, como por ejemplo, el receptor seleccionado. Es decir, si el usuario selecciona el receptor 2 en la pantalla 2 (RXS) desde *screen02Actions()*, enviamos a las pantallas 3 y 4, que el receptor seleccionado en este momento, es el 2.

```
void Chelton_SESClass::Screen01Actions ()
{
  if ( 1 == actualScreen){

  }


}
void Chelton_SESClass::Screen02Actions ()
{

  if ( 2 == actualScreen){

  //std::cout << "Screen02Action: " << (int) actualScreen << std::endl;

  if (( 1 == keyUpGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10)
  {
    display->cheltonDisplay02->upPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay02->upPressed = false;

  if (( 1 == keyDownGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 1
  {
    display->cheltonDisplay02->downPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay02->downPressed = false;

  if (( 1 == keyLeftGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10
  {
    display->cheltonDisplay02->leftPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay02->leftPressed = false;

  if (( 1 == keyRightGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 1
  {
    display->cheltonDisplay02->rightPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay02->rightPressed = false;

  if ( 1 == keyEnterGlsPushButton->State() )
    display->cheltonDisplay02->enterPressed = true;
  else
    display->cheltonDisplay02->enterPressed = false;



  //display->cheltonDisplay02->rxSelected = display->cheltonDisplay14->rxSelected;
```

```
    display->cheltonDisplay03->rxSelected = display->cheltonDisplay02->rxSelected;
    display->cheltonDisplay04->rxSelected = display->cheltonDisplay02->rxSelected;


    /* The selected channel in display03, is set in display04 and display 05*/
    display->cheltonDisplay03->channelSelected = display->cheltonDisplay02->channelSelected;
    display->cheltonDisplay04->channelSelected = display->cheltonDisplay02->channelSelected;


  }

}
void Chelton_SESClass::Screen03Actions ()
{
  if ( 3 == actualScreen){

  if (( 1 == keyUpGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10)
  {
    display->cheltonDisplay03->upPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay03->upPressed = false;

  if (( 1 == keyDownGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 1
  {
    display->cheltonDisplay03->downPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay03->downPressed = false;

  if (( 1 == keyLeftGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
    display->cheltonDisplay03->leftPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay03->leftPressed = false;

  if (( 1 == keyRightGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
    display->cheltonDisplay03->rightPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay03->rightPressed = false;


  if ( 1 == key3Pressed )
    display->cheltonDisplay03->key3Pressed = true;
  else
    display->cheltonDisplay03->key3Pressed = false;


    /* The selected receiver in display03, is set in display04 and display 05*/
```

```
    display->cheltonDisplay02->rxSelected = display->cheltonDisplay03->rxSelected;
    display->cheltonDisplay02->selectReceiverGlsSwitch->DetentVal(display->cheltonDisplay02->rxSelec
    display->cheltonDisplay04->rxSelected = display->cheltonDisplay03->rxSelected;


    /* The selected channel in display03, is set in display04 and display 05*/
    display->cheltonDisplay02->channelSelected = display->cheltonDisplay03->channelSelected;
    display->cheltonDisplay04->channelSelected = display->cheltonDisplay03->channelSelected;


  }
}
void Chelton_SESClass::Screen04Actions ()
{
  if ( 4 == actualScreen){

  if (( 1 == keyUpGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
    display->cheltonDisplay04->upPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay04->upPressed = false;

  if (( 1 == keyDownGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
    display->cheltonDisplay04->downPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
    display->cheltonDisplay04->downPressed = false;

  if (( 1 == keyLeftGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
    display->cheltonDisplay04->leftPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay04->leftPressed = false;

  if (( 1 == keyRightGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
    display->cheltonDisplay04->rightPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay04->rightPressed = false;


    /* The selected receiver in display04, is set in display03 and display 05*/
    display->cheltonDisplay02->rxSelected = display->cheltonDisplay04->rxSelected;
    display->cheltonDisplay02->selectReceiverGlsSwitch->DetentVal(display->cheltonDisplay02->rxSelec
    display->cheltonDisplay03->rxSelected = display->cheltonDisplay04->rxSelected;


    /* The selected channel in display04, is set in display03 and display 05*/
    display->cheltonDisplay02->channelSelected = display->cheltonDisplay04->channelSelected;
    display->cheltonDisplay03->channelSelected = display->cheltonDisplay04->channelSelected;
```

```
    }

}
void Chelton_SESClass::Screen05Actions ()
{
    if ( 5 == actualScreen){

    if ( 1 == key5GlsPushButton->State() )
        display->cheltonDisplay05->key5Pressed = true;
    else
        display->cheltonDisplay05->key5Pressed = false;

    if ( 1 == key6GlsPushButton->State() )
        display->cheltonDisplay05->key6Pressed = true;
    else
        display->cheltonDisplay05->key6Pressed = false;

    if ( 1 == key12GlsPushButton->State() )
        display->cheltonDisplay05->key12Pressed = true;
    else
        display->cheltonDisplay05->key12Pressed = false;

    }

}
void Chelton_SESClass::Screen06Actions ()
{
    if ( 6 == actualScreen){


    }

}
void Chelton_SESClass::Screen07Actions ()
{
    if ( 7 == actualScreen){


    }

}
void Chelton_SESClass::Screen08Actions ()
{
    if ( 8 == actualScreen){


    }

}
```

```
void Chelton_SESClass::Screen09Actions ()
{
  if ( 9 == actualScreen){

  if (( 1 == keyUpGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
    display->cheltonDisplay09->upPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay09->upPressed = false;

  if (( 1 == keyDownGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
    display->cheltonDisplay09->downPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay09->downPressed = false;


  }

}
void Chelton_SESClass::Screen10Actions ()
{
  if ( 10 == actualScreen){



  }

}
void Chelton_SESClass::Screen11Actions ()
{
  if ( 11 == actualScreen){



  }

}
void Chelton_SESClass::Screen12Actions ()
{
  if ( 6 == actualScreen){



  }

}
void Chelton_SESClass::Screen13Actions ()
{
  if ( 13 == actualScreen){
```

```
    }

}
void Chelton_SESClass::Screen19Actions ()
{
  if ( 19 == actualScreen){

  if (( 1 == keyUpGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
     display->cheltonDisplay19->upPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay19->upPressed = false;


  if (( 1 == keyDownGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
     display->cheltonDisplay19->downPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay19->downPressed = false;


  if (( 1 == keyLeftGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
     display->cheltonDisplay19->leftPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay19->leftPressed = false;

  if (( 1 == keyRightGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
     display->cheltonDisplay19->rightPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay19->rightPressed = false;


  if ( 1 == key5GlsPushButton->State() )
     display->cheltonDisplay19->key5Pressed = true;
  else
     display->cheltonDisplay19->key5Pressed = false;

  if ( 1 == key6GlsPushButton->State() )
     display->cheltonDisplay19->key6Pressed = true;
  else
     display->cheltonDisplay19->key6Pressed = false;


  if ( 1 == key12GlsPushButton->State() )
     display->cheltonDisplay19->key12Pressed = true;
  else
```

```
    display->cheltonDisplay19->key12Pressed = false;


  if ( 1 == key13GlsPushButton->State() )
    display->cheltonDisplay19->key13Pressed = true;
  else
    display->cheltonDisplay19->key13Pressed = false;

  }

}
void Chelton_SESClass::Screen18Actions ()
{
  if ( 18 == actualScreen){

  if (( 1 == keyUpGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
    display->cheltonDisplay18->upPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay18->upPressed = false;


  if (( 1 == keyDownGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
    display->cheltonDisplay18->downPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay18->downPressed = false;


  if (( 1 == keyLeftGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
    display->cheltonDisplay18->leftPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay18->leftPressed = false;

  if (( 1 == keyRightGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
    display->cheltonDisplay18->rightPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay18->rightPressed = false;


  }

}
void Chelton_SESClass::Screen17Actions ()
{
  if ( 17 == actualScreen){
```

```
  if (( 1 == keyUpGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
    display->cheltonDisplay17->upPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay17->upPressed = false;


  if (( 1 == keyDownGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
    display->cheltonDisplay17->downPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay17->downPressed = false;


  if (( 1 == keyLeftGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
    display->cheltonDisplay17->leftPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay17->leftPressed = false;

  if (( 1 == keyRightGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
    display->cheltonDisplay17->rightPressed = true;
    lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
    display->cheltonDisplay17->rightPressed = false;


  if ( 1 == key6GlsPushButton->State() )
    display->cheltonDisplay17->key6Pressed = true;
  else
    display->cheltonDisplay17->key6Pressed = false;

  if ( 1 == key12GlsPushButton->State() )
    display->cheltonDisplay17->key12Pressed = true;
  else
    display->cheltonDisplay17->key12Pressed = false;

  display->cheltonDisplay05->rxSelected = display->cheltonDisplay17->MBRType+1;

  }

}
void Chelton_SESClass::Screen16Actions ()
{
  if ( 16 == actualScreen){
```

```
  if (( 1 == keyUpGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
     display->cheltonDisplay16->upPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay16->upPressed = false;


  if (( 1 == keyDownGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
     display->cheltonDisplay16->downPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay16->downPressed = false;


  if (( 1 == keyLeftGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
     display->cheltonDisplay16->leftPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay16->leftPressed = false;

  if (( 1 == keyRightGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
     display->cheltonDisplay16->rightPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay16->rightPressed = false;

  if ( 1 == keyEnterGlsPushButton->State() )
     display->cheltonDisplay16->key9Pressed = true;
  else
     display->cheltonDisplay16->key9Pressed = false;


  }

}
void Chelton_SESClass::Screen15Actions ()
{
  if ( 15 == actualScreen){



  }

}
void Chelton_SESClass::Screen14Actions ()
{
  if ( 14 == actualScreen){
```

```
  if (( 1 == keyUpGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
     display->cheltonDisplay14->upPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay14->upPressed = false;


  if (( 1 == keyDownGlsPushButton->State() ) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
     display->cheltonDisplay14->downPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay14->downPressed = false;


  if (( 1 == keyLeftGlsPushButton->State()) || (cheltonPowerGlsKnob->PositionVal() > lastPosition + 10))
  {
     display->cheltonDisplay14->leftPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay14->leftPressed = false;

  if ((1 == keyRightGlsPushButton->State()) || (cheltonPowerGlsKnob->PositionVal() < lastPosition - 10))
  {
     display->cheltonDisplay14->rightPressed = true;
     lastPosition = cheltonPowerGlsKnob->PositionVal();
  }
  else
     display->cheltonDisplay14->rightPressed = false;


  if ( 1 == keyEnterGlsPushButton->State() )
     display->cheltonDisplay14->enterPressed = true;
  else
     display->cheltonDisplay14->enterPressed = false;


  display->cheltonDisplay14->rxSelected = display->cheltonDisplay02->rxSelected;

  }

}
```

3.2.2. CONEXIÓN ENTRE EL MÓDULO Y LA PARTE GRÁFICA

- La parte gráfica y el módulo se conectan mediante puertos UDP, recogen la ip y los puertos de cliente – servidor respectivamente, y así se mandan información de uno a otro.

3.3. FASE 3: PRUEBAS EN APARATO FÍSICO

Una vez realizado el módulo y la parte gráfica, el simulador debe probarse en el simulador general del helicóptero en las instalaciones de INDRA San Fernando para comprobar que realmente funciona todo correctamente.

Después de una primera visita al simulador el 27 de Enero, se comprobó que lo que estaba hasta ese momento creado, funcionaba correctamente. A pesar de ello, había una serie de errores de hardware en el aparato, como por ejemplo que, el potenciómetro, tenía un tope y había que quitarlo.

En una segunda visita a primeros de Abril, se volvieron a realizar más pruebas, encontrando un error de hardware – software que consistía en que, con cualquiera de los botones del Chelton, se recibía la orden de apagado en el módulo, se acabó subsanando correctamente. Además de esto, hubo un problema con el tamaño del monitor en el que se visualiza, ya que, en la estructura en la que estaba el tft del Chelton insertado, tapaba cerca de 2 centímetros la pantalla, quitando así espacio para visualizarlo.

En una tercera visita, se realizaron las pruebas oficiales del Chelton, explicadas en el ANEXO A de este documento. La ATP son las pruebas oficiales del módulo, en este caso, del Chelton, pero que deben pasar todos los distintos módulos creados en el simulador.

CONCLUSIONES

Problemas encontrados durante el desarrollo del proyecto

Algunos de los problemas encontrados durante el desarrollo del proyecto son los siguientes:

- Desconocimiento del funcionamiento del aparato real: Al no tener un aparato con el que hacer las pruebas, hemos tenido que guiarnos por la lógica del funcionamiento del aparato (pensar que algo se hace de una manera porque parece lo más razonable, pero no tener la certeza de que es así).
- Grave problema a la hora de tratar el receptor 1: El receptor 1, contiene 10 canales. Inicialmente, tratábamos alguna excepción del receptor 1, pero finalmente, debido a que esos 10 canales que posee, representan realmente 10 receptores nuevos, hemos tenido que añadirlo en las estructuras como tal, y cambiar todo el código prácticamente al final del desarrollo del módulo.
- Conexión entre parte gráfica y módulo: Otra de los problemas fue enlazar ambas partes, ya que, deben contener una estructura común y una serie de requisitos para que funcionen a la vez.
- Malas interpretaciones en las reuniones con el jefe: A medida que surgían dudas sobre el funcionamiento del aparato, se hizo una serie de preguntas al jefe de proyecto, y en algunos casos, obteníamos distintas respuestas en fechas distintas, teniendo que cambiar 2 veces un mismo método.

Conclusiones Finales

Después de realizar durante 9 meses un proyecto en la empresa INDRA, he adquirido una nueva experiencia que más adelante ayudara a cometer menos errores que los cometidos en este proyecto. Cabe destacar los siguientes puntos:

- Aprendizaje del funcionamiento interno de la empresa: Durante los primeros dos meses, aprendí a hacer funcionar el simulador, estudié la estructura del simulador, conocí cómo funcionaba y era capaz de hacerlo funcionar.
- Conocimiento de los distintos protocolos internos de la empresa: La empresa INDRA, al ser tan grande, dispone de una serie de protocolos a seguir, en cuanto a la creación de nuevos códigos, para que en un futuro, sea más sencillo subsanar errores. Estos protocolos incluyen, documentos oficiales necesarios, estructuras de código que deben ser de una forma determinada, utilización de diferentes herramientas de trabajo estandarizadas, etc.

- La importancia de una buena organización por parte del jefe de proyecto y un buen trabajo en equipo: Una de las cosas que más he valorado en la finalización del proyecto, ha sido, ver cómo el trabajo realizado en el ordenador personal del que disponía en mi lugar de trabajo en el CES en León, funcionaba en el simulador físico , en Madrid, gracias a que, un equipo encargado del Hardware, otro encargado de recoger la información del hardware y mandar la información a mi módulo, y todo ello sin estar en constante contacto con los otros departamentos.

Posibles mejoras del sistema

Debido al contrato del simulador, el CHELTON DF935 no tenía todas las funcionalidades reales implementadas, así que las posibles mejoras son las siguientes:

- Inclusión de la modalidad de mensajes: Se podría implementar toda la parte de código que conllevase hacer funcionar la recepción de mensajes de tipo DSC y SARSAT.
- Inclusión de la funcionalidad del botón TEST: Podríamos crear una serie de métodos que testearan al completo el aparato, para comprobar que no tiene ningún tipo de errores.
- Creación de método que reciba un mayor número de balizas: Actualmente el aparato recibe información de 5 balizas, podría crearse un procedimiento que permitiera recibir más balizas.

LISTA DE REFERENCIAS

[1] Manual de operaciones del Chelton 935-11-600

[2] Manual del *EC225 - DDI_08020A*

[3] Manual de LCD Simulado disponible en www.purdyelectronics.com

[4] Manual de usuario *de EL Small Graphics Display* USER'S MANUAL

[5] Manual de glStudio disponible en http://www.disti.com/releases/index.html

[6] Manual de programación C++ disponible en: http://www.sisoft.ucm.es/Manuales/C++.pdf

ANEXO A

PROGRAM:	**EC225 FFS LEVEL B**
TITLE:	**ATP: NORMAL PROCEDURES TEST CHAPTER (CHELTON DF 935)**

	NAME	SIGNATURE	DATE
PREPARED	Design Groups		
REVIEWED	Mª Eva Sánchez Jiménez		
QUALITY ASSURANCE	C. Vargas Hilla		
AUTHORIZED	F. Maderuelo Pérez		

DISTRIBUTION LIST

EXTERNAL	INTERNAL
A. Eurocopter (PDF file)	**B. Configuration Management** (MS Word, PDF files) **C. Local Network** (Word and PDF files)

DOCUMENT CHANGE RECORD

Edition	Date	Reason for Change
Draft	21/05/2010	Initial Draft

LIST EFFECTIVE PAGES

The present document is composed of the following pages:

For a total of 41 pages.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. PURPOSE

This document contains the Visual Inspection Tests, part of the Acceptance Test Protocols for the EC225 FFS Level B simulator.
The purpose of this document is to establish an agreed, objective rational to demonstrate that the EC225 training device delivered under this contract fulfils the requirements of the contract, as well as the requirements of the applicable FAA / JAR standard for qualification.

1.2. SCOPE AND APPLICABILITY

This document, of reference '0915700000000AT05' and title 'ATP: NORMAL PROCEDURES TEST CHAPTER' applies to the activities, tasks and works required for the execution of the Program 'EC225 FFS Level B' requested by Eurocopter (EC).

The 'EC225 FFS LEVEL B' program will provide EC with the required means for effective training of the EC225 helicopter aircrews.

Items and services, object of this Program, to be developed and delivered by Indra are as follow:

- One (1) Full Flight Simulator (FFS), (herein after referred to as "FFS-001") to be delivered to EC in Aberdeen (UK).

1.3. ACRONYMS AND ABBREVIATIONS

1.3.1. DEFINITIONS

Acceptance. An action by an authorized representative of the acquirer by which the acquirer assumes ownership of the products as partial or complete performance of a contract.

Acceptance Test. The acceptance test shall demonstrate the serviceability of the equipment on receipt by the purchaser. It shall ensure that the equipment meets the build standard, quality and basic function/performance required by the specification.

Agreement. The definition of terms and conditions under which a working relationship will be conducted.

Approval. The agreement that an item is complete and suitable for its intended use.

Architecture. The organisational structure of a system, CSCI or HWCI, identifying its components, their interfaces, and a concept of execution among them.

Build. A version of the system that meets a specified subset of the requirements that the completed system will meet.

Built-In-Test (BIT). The hardware and software facilities integrated into a system, sub-system, equipment or LRI or module to monitor functions and check out serviceability.

Client. Organization or person that receives the product.

Commercial-Off-The-Shelf (COTS). A non-developmental item that has been produced for sale in the commercial marketplace.

Computer program. A combination of computer instructions and data definitions that enable computer hardware to perform computational or control functions.

Computer Software Configuration Item (CSCI). An aggregation of software that satisfies an end use function and is designated for separate configuration management by the acquirer. CSCIs are selected based on tradeoffs among software function, size, host or target computers, developer, support concept, plans for reuse, criticality, interface considerations, need to be separately documented and controlled, and other factors.

Configuration Item (CI). An aggregation of hardware, software, or both that satisfies an end use function and is designated for separate configuration management by the acquirer.

Design. Those characteristics of a system, HWCI or CSCI that are selected by the developer in response to the requirements. Some will match the requirements; others will be elaborations of requirements, such as definitions of all error messages in response to a requirement to display error messages; others will be implementation related, such as decisions of what software units and logic to use to satisfy the requirements.

Developer. An organisation that develops products. “Develops” may include new development, modification, reuse, reengineering, maintenance, or any other activity that results in products.

Elicitation. The process of capturing and discovering of requirements.

Evaluation. The process of determining whether an item or activity meets specified criteria.

Firmware. The combination of a hardware device and computer instructions and/or computer data that reside as read-only software on the hardware device.

Hardware. Material made items and their components (mechanical, electrical, electronic, hydraulic, pneumatic). Computer software and technical documentation are excluded.

Hardware Configuration Item (HWCI). An aggregation of hardware that satisfies an end use function and is designated for separate configuration management by the acquirer.

Interface. The required features that exist in a shared borderline. In development, a relationship among two or more entities (such as CSCI-CSCI, CSCI-HWCI, HWCI-HWCI, HWCI-User, CSCI-User or software unit to software unit) where the entities share, provide or exchange data. An

interface is not a CSCI, HWCI, software unit or other system component; it's a relationship among them.

Life cycle model. A framework containing the processes, activities, and tasks involved in the development, operation, and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use.

Operator. An individual or organisation that operates the system.

Process. A set of interrelated activities, which transform inputs into outputs.

Reengineering. The process of examining and altering an existing system to reconstitute it in a new form. May include reverse engineering (analysing a system and producing a representation at a higher level of abstraction, such as design from code), restructuring (transforming a system from one representation to another at the same level of abstraction), redocumentation (analysing a system and producing user or support documentation), forward engineering (using software products derived from an existing system, together with new requirements, to produce a new system), retargeting (transforming a system to install it on a different target system), and translation (transforming source code from one language to another or from one version of a language to another).

Requirement. A characteristic that a system, HWCI or CSCI must possess in order to be acceptable by the acquirer.

Reusable software product. A software product developed for one use but having other uses, or one developed specifically to be usable on multiple projects or in multiple roles on one project. Examples include, but are not limited to, commercial off-the-shelf software products, acquirer-furnished software products, software products in reuse libraries, and pre-existing developer software products. Each use may include all or part of the software product and may involve its modification. This term can be applied to any software product (for example, requirements, architectures, etc.), not just to software itself.

Software. Computer programs and computer databases.

Software development. A set of activities that results in software products. Software development may include new development, modification, reuse, reengineering, maintenance, or any other activities that result in software products.

Software development process. An organized set of activities performed to translate user needs into software products.

Software product. Software or associated information created, modified, or incorporated to satisfy a contract. Examples include plans, requirements, design, code, databases, test information, and manuals.

Software unit. An element in the design of a CSCI; for example, a major subdivision of a CSCI, a component of that subdivision, a class, object, module, function, routine, or database. Software units may occur at different levels of a hierarchy and may consist of other software units. Software units in the design may or may not have a one-to-one relationship with the code and data entities (routines, procedures, databases, data files, etc.) that implement them or with the computer files containing those entities.

System. A self-sufficient unit in its intended operational environment, which includes all equipment, related facilities, material, software, services, and personnel required for its operation and support..

Traceability. The ability to relate an input to a phase of the system or software life cycle to an output from that phase. The item may be software or documentation.

Use Case. A technique to enhance the understanding of the requirements that is elaborated alongside with the requirements specification. It describes in narrative form the event sequence of an actor (agent external to the system) that uses the system to perform a certain process. Each use case is a possible way of using the system. A use case is a complete process.

User. An individual or organisation that uses the operational system to perform a specific function.

Validation. A process to guarantee that the right product is built, namely, that the final product fulfils the intended use.

Verification. A process to guarantee that the product is properly built, namely, to determine that the products of an activity fulfil the requirements of the previous activity.

1.3.2. ACRONYMS AND ABREVIATIONS

AA	Airworthiness Authorities.
ATP	Acceptance Test Plan
ATP	Acceptance Tests Procedures.
BFE	Buyer Furnished Equipment
BFI	Buyer Furnished Item
CDB	Common Data Base
CDR	Critical Design Review.
CDRL	Contract Data Requirement List
CGF	Computer Generated Forces
CLS	Control Loading System
CMMi	Capability Maturity Model integration.
COTS	Commercial Off The Shelf.
CR	Completion Review
DACS	Digital Audio and Communication System
DF	Debriefing
DIL	Deliverables Item List
DP	Data Package
EASA	European Aviation Safety Agency
EC	Eurocopter
ED	Electronic Device
EFA	Engineer Functional Assessment
EGPW	Enhanced Ground Proximity Warning System
FAT	Factory Acceptance Tests.
FFS	Full Flight Simulator
FIR	Final Installation Review
FLIR	Forward Light Infra Red
FLM	Flight Manual
FMECA	Failure Mode, Effects, and Criticality Analysis

FSTD	Flight Synthetic Training Device
FTD	Flight Training Device
H/C	Helicopter
HLA	High Level Architecture
HMI	Human Machine Interface
HW	HardWare.
IFR	Instrumental Flight Rules.
IG	Image Generator
ILS	Integrated Logistic Support.
IOS	Instructor Operation Station
ISMS	Integrated Simulation Management System
ITP	Instruction to Proceed.
JAA	Joint Aviation Authorities
JAR	Joint Aviation Requirements.
JWG	Joint Working Group
MCC	Multi-Crew Coordination.
MoM	Minutes of Meeting
MS	Motion System
OSAT	On-Site Acceptance Tests.
PCA	Physical Configuration Audit.
PDF	Portable Document Format.
PDR	Preliminary Design Review.
PSA	Pilot Subjective Assessment
QA	Quality Assurance.
QTG	Qualification Test Guide
RAD	Requirements Analysis Document
RFT	Ready for Training
RTA	Real Time Architecture
RTM	Requirements Traceability Matrix
RFU	Ready for Use
SAR	Search and Rescue
SE	System Engineering
SEMP	System Engineering Management Plan.
SFEAT	System requirements
SOW	Statement of Work
SRR	System Requirements Review.
SSL	Simulación y Sistemas Logísticos (*Simulation and Support Systems*)
STD	Synthetic Training Device.
SW	SoftWare.
TCAS	Traffic Collision Avoidance System
TRR	Test Readiness Review.
T&TE	Tools and Test Equipment
UK	United Kingdom
VDB	Visual Data Base
VDS	Visual Display System
VFR	Visual Flight Rules.

VRR	Validation Readiness Review
WBS	Work Breakdown Structure.
WGM	World Geodetic Model

PAGE INTENTIONALLY LEFT BLANK

2. DOCUMENTS

2.1. APPLICABLE DOCUMENTS

EC255 FFS Contract:

- INDRA / EC contract for delivery of one EC225 FFS Level B.
- Appendix B2_ETCHD_023_B (EC225 FTD - Eurocopter FSDP SOW).pdf
- JAR FSTD-H.

Program Plans:

- Program Management Plan (0915700000000PG00).
- Requirements Traceability Matrix (0915700000000PT00).
- Logistic Support Plan (0915700000000PL00).
- Quality Assurance Plan (0915700000000QA00).
- Acceptace Test Plan (0915700000000TP00).

Design Documents:

- Requirements Traceability Matrix (RTM, 0915700000000RH00).
- System and Subsystem Design Document (SSDD, 0915700000000DF04).

Program Plans (Indra internal Plans):

- System Engineering Plan (0915700000000PG01).
- Risk Analysis Plan (0915700000000RA00).
- Documentation Plan (0915700000000CG00).
- Configuration Management Plan (0915700000000CP00).
- Packing and Shipping Plan (0915700000000EM00).

2.2. REFERENCE DOCUMENTS

Indra internal procedures:

• IP-GS-0310	Guía para la Preparación de WBS *(WBS Preparation Guide)*
• MAN-IDR-130	Método Indra de Gestión de Proyectos *(Project Management Method for Indra)*

- PPC-GS-607 Estimación de Tamaño y Esfuerzo Sw *(Sw Size and Effort Estimation)*
- PPC-ID-944 Revisiones por iguales *(Peer Reviews)*
- PPC-GS-550 Estimación de Costes Directos SIMSAM *(SIMSAM Direct Costs Estimation)*
- PPC-ID-944 Revisión por iguales (*Peer Reviews*)
- PPC-ID-602 Planificación, seguimiento y control de proyectos *(Project Planning, Monitoring and Control)*
- PPC-GS-606 Seguimiento por la Dirección *(Reporting to Higher Management)*
- PPC-GS-607 Estimación SW (*Estimations of software size and effort*)
- PPC-ID-942 Verificación y validación *(Validation and Verification)*
- PPC-ID-943 Revisiones Técnicas *(Technical Reviews)*
- PPC-ID-945 Ingeniería de Requisitos *(Requirements Engineering)*
- PPC-ID-946 Ingeniería de Sistemas *(Systems Engineering)*
- PPC-ST-567 Ingeniería Hardware *(Hardware Engineering)*
- PPC-ST-568 Ingeniería Software *(Software Engineering)*
- PRP-IDR-080 Gestión de Configuración *(Configuration Management)*
- PPC-ID-600 Gestión de Riesgos *(Risk Management)*
- PRG-IDR-171 Guía para la Formación adicional *(Guide for Additional Training)*
- IG-IDR-073 Gestión de Compras y Subcontrataciones, Actividades Complementarias Áreas CMMi *(Procurement and Subcontracting Management)*
- IP-ID-6900 Guía para la Adaptación de Procesos y Productos *(Process and Products Tailoring Guide)*
- PRG-IDR-350 Marcado CE *(EC marking)*

In case of conflict, the contract documentation will have precedence above any other document.

3. REQUIREMENTS TRACEABILITY

The following requirements are verified in this test:

4. TEST CASES

This document contains the following test cases:

- FFS-NPR-001: CHECKS for Chelton DF 935

5. FFS-NPR-001: CHECKS FOR CHELTON DF 935.

ACCEPTANCE TEST PROCEDURES FOR EC225 FFS Level B	
TEST NUMBER	**TEST NAME:**
FFS-NPR-001	Checks for Chelton DF 935

5.1. REQUIRED ROLES/SKILLS FOR PERFORMING THIS TEST

- ☒ Qualified test pilot.
- ☒ Flight engineer.
- ☐ Simulator instructor.
- ☐ Simulator technician.
- ☐ No special skill need.

Note: marked profiles are qualified for carrying out this test during final acceptance.

5.2. PURPOSE OF THE TEST:

The purpose of this test is checking the Chelton DF 935 functionality.

5.3. TEST REQUIREMENTS:

The Simulator itself.
Flight Manual copy or extract.
Chelton Operational Manual

5.4. REQUIRED MEANS

None

5.5. TEST PROCEDURE

5.5.1. Initial conditions

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
1.	Cockpit: check cockpit status is compatible with "**BOTH ENGINES OFF**" condition	Cockpit OFF	
2.	IOS: load mission "**ATP TEST 3**"	Mission is recalled	

5.5.2. Operation

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
1.	On cockpit: Active the battery	Chelton has been supplied.	
	POWER UP		
2.	Press and hold the ON/OFF button until the screen illuminates (approximately 3 seconds).	The display initially shows the Splash Screen, which indicates that the unit is initialising the system. 	

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
		After a period of less than 5 seconds, the display changes to the RXR STATUS screen:  The system always starts with the Guard Receivers selected to their MAIN channel (“M” on the picture on top).	
	POWER DOWN		
3.	Press and hold the ON/OFF button until the screen blanks (approximately 3 seconds).	The following display is shown in Chelton: 	
	FACTORY SETTINGS		
4.	Press the RIGHT key while switching the controller on (either by applying power to the system or, if power already applied, by holding the ON/OFF button active until the screen illuminates)	The screen displays “SETTING FACTORY DEFAULTS” for less than 6 seconds: 	

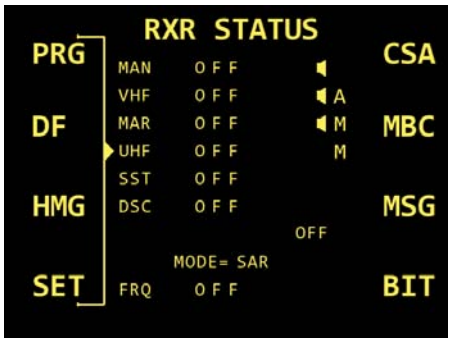
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
	MONITORING OPERATION		
5.	Select the receiver by rotating the knob or pressing the UP/DOWN navigation key.	The arrow points to the required receiver: 	
6.	If any active receiver detects a signal.	The receiver is shown highlighted in the list: 	
7.	If RECEIVER 1 is selected, press the ENTER key.	The display toggles between the channel number and the frequency in that channel: 	

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
			
8.	If RECEIVER 1 is selected, press the LEFT/RIGHT navigation key.	The selected channel is incremented or decremented.   	

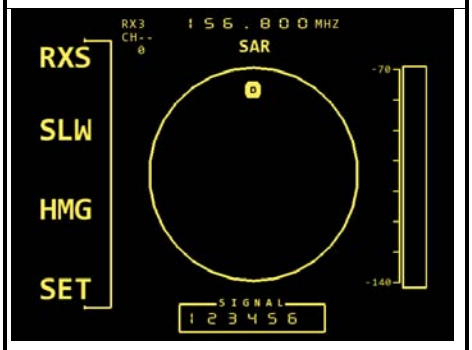
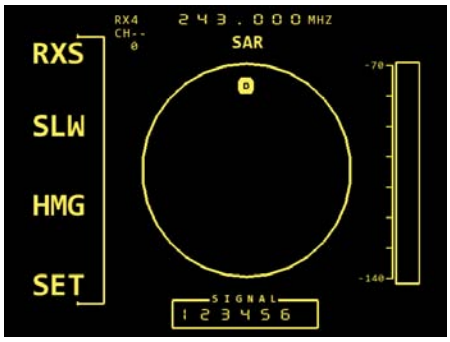
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
9.	If RECEIVER 2, 3 or 4 is selected, press the ENTER key.	The receiver toggles between its MAIN (“M”) and AUXILIARY (“A”) frequency.   	
10.	If RECEIVER 5 or 6 is selected, this screen has no adjustable selections.	No results. Receiver 5 selected:  	

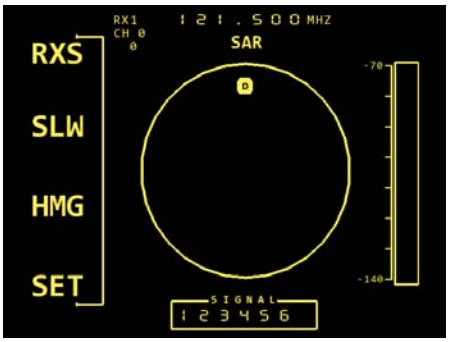
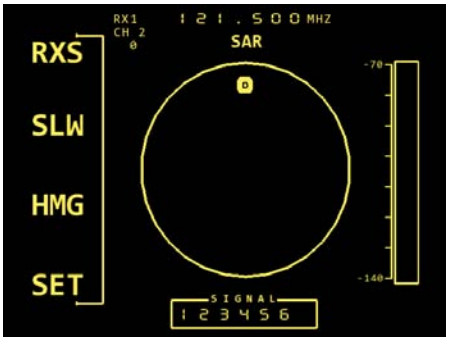
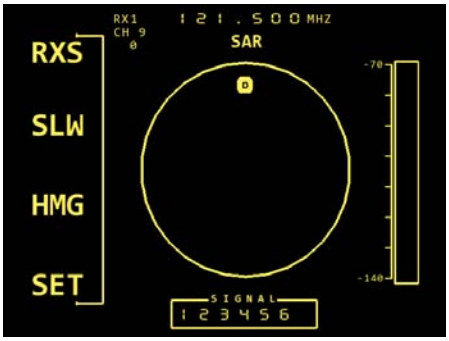
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
		Receiver 6 selected: 	
11.	Set a beacon activated in each receiver.	The receivers are shown highlighted in the list:  The beacon data is shown in DF or HMG screen for each receiver: 	

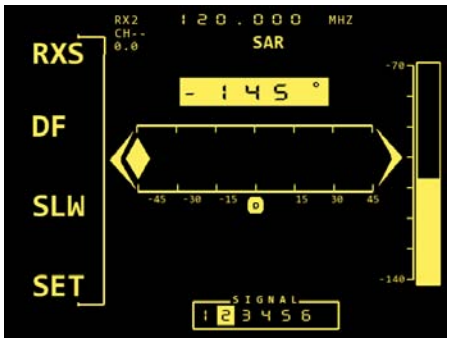
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
12.	Change the beacon's position or the helicopter's position.	The bearing data and the signal strength change:    	
13.	Press the PRG soft-key, then select the ON/OFF option in PROG RX screen and press the ENTER key. Do it with all receivers.	The receiver toggles between ON/OFF:  	

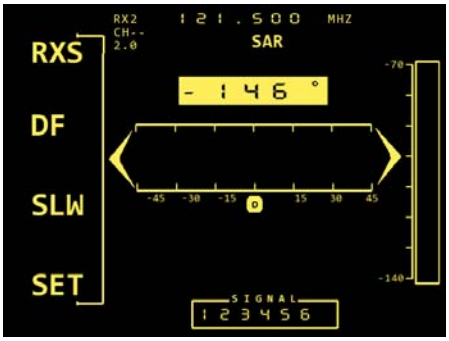
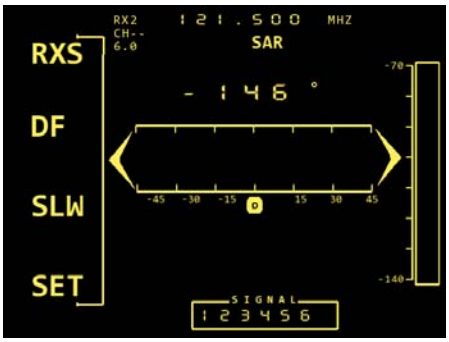
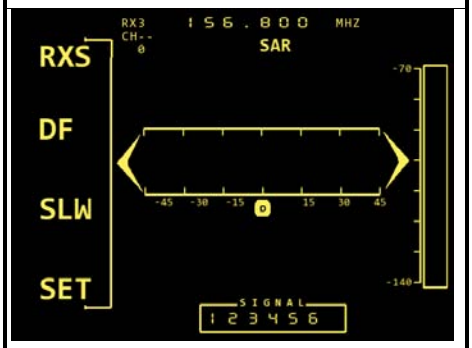
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
		Now, the receivers show “OFF” instead of the frequency: 	
14.	Press the PRG soft-key, then select AUDIO option in PROG RX screen and press ENTER key.	The receiver toggles between NORMAL/MUTE (for example RX4):  	

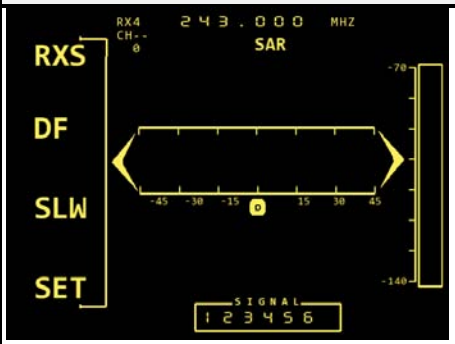
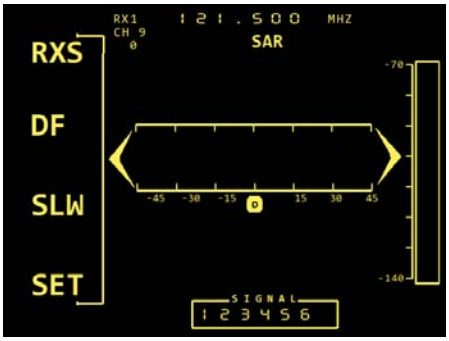
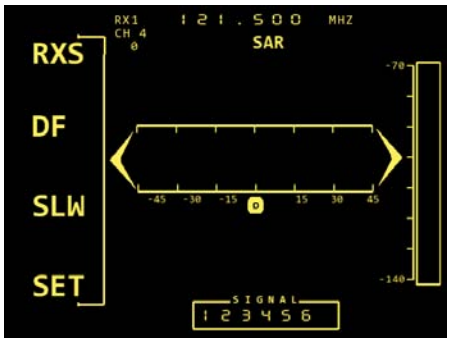
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
		Then the receiver' loudspeaker icon disappears in this RXR STATUS screen: 	
	DF OPERATION		
15.	Select a receiver in the RXR STATUS screen and press the DF soft-key.	The display changes to the DF screen: 	
16.	If the signal on the active channel falls below the threshold at which the DF is able to take a bearing.	The bearing display is frozen and the age displayed is incremented: 	

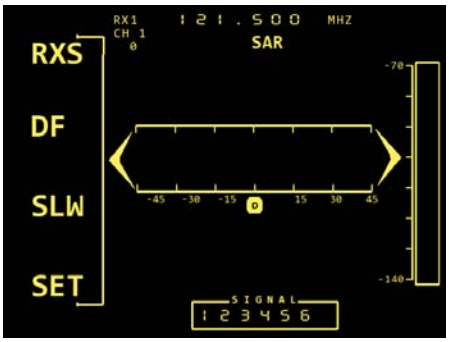
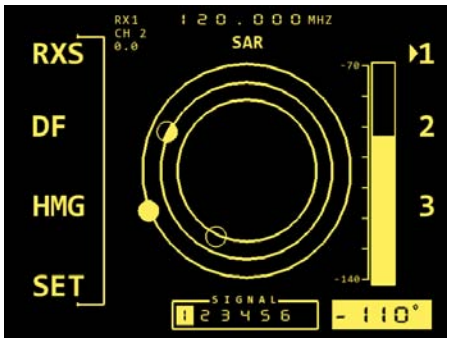
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
17.	If the age displayed exceed 4 seconds.	The bearing is no longer highlighted, but the last known value is shown. 	
18.	Press the UP/DOWN navigation key	Increases/decreases the receiver number:  	

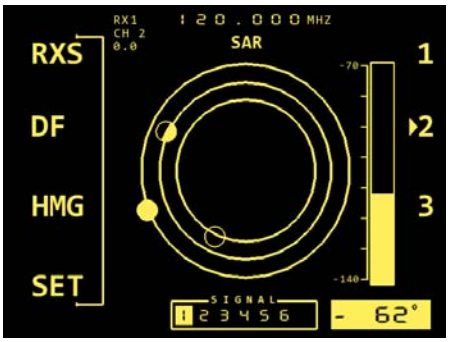
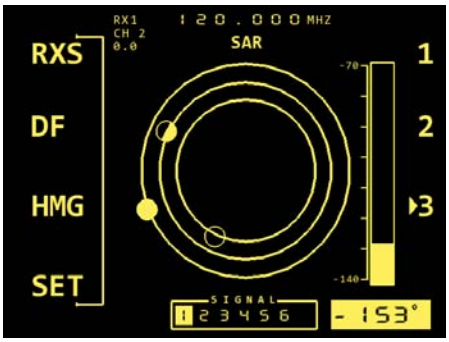
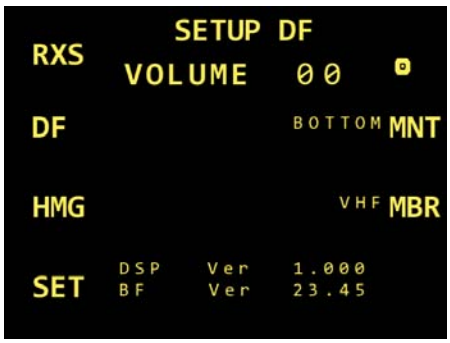
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
19.	Select the RECEIVER 1 and press the LEFT/RIGHT navigation key	Changes the receiver's channel: Channel 0 is shown:  Channel 2 is shown:  Channel 9 is shown:  	

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
20.	Press the SLW soft-key. 	Enables the SLEW function and rotating the knob we force the displayed bearing angle to move: 	
21.	Release the SLW soft-key	The display returns to show bearing to target only if a valid bearing is being detected.	
	HOMING OPERATION		
22.	Select a receiver in the RXR STATUS screen and then press the soft-key HMG.	The display changes to the HMG screen: 	

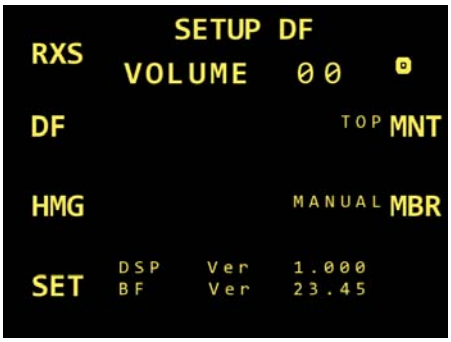
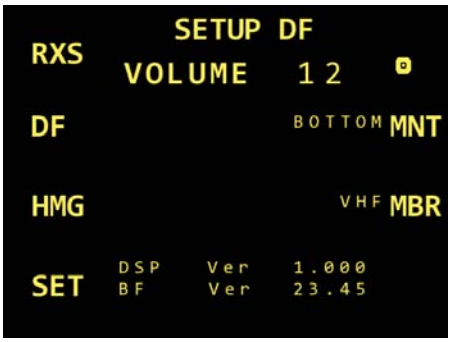
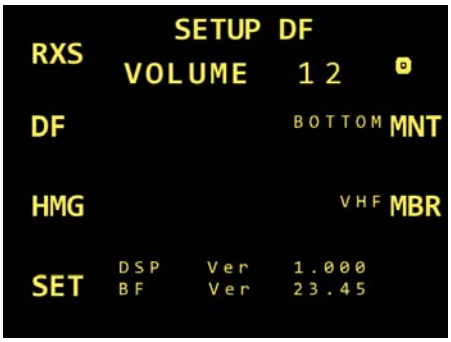
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
23.	If the signal on the active channel falls below the threshold at which the DF is able to take a bearing.	The bearing display will be frozen and the age displayed is incremented. 	
24.	If the age displayed exceed 4 seconds.	The bearing is no longer highlighted, but the last known value is shown. 	
25.	Press the UP/DOWN navigation key	Increases/decreases the receiver number: 	

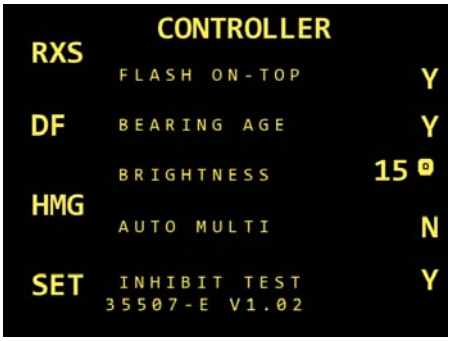
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
			
26.	Select the RECEIVER 1 and press the LEFT/RIGHT navigation key	Changes the receiver's channel: Channel 9 is shown:  Channel 4 is shown: 	

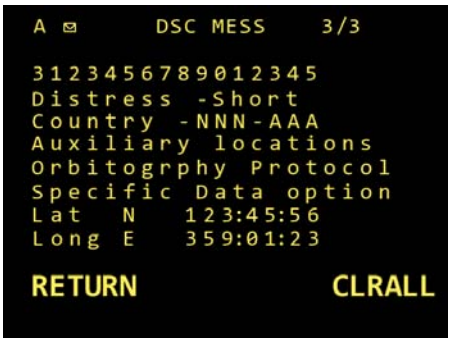
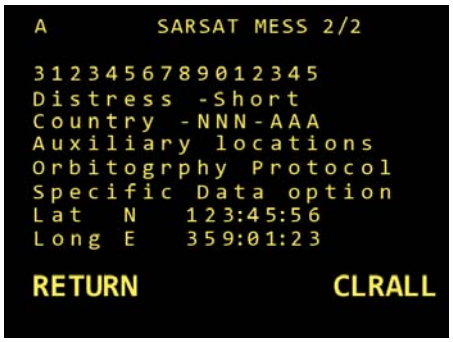
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
		Channel 1 is shown: 	
27.	Press the SLW soft-key.	Enables the SLEW function and forces the displayed bearing angle to move.	
28.	Release the SLW soft-key	The display returns to show bearing to target only if a valid bearing is being detected.	
	MULTI-BEACON OPERATION		
29.	Press MBC soft-key and then press the soft-key adjacent to the numbers 1, 2 and 3 on the right of the screen.	The display shows the age, signal strength and numeric bearing details of beacon selected. Data of beacon 1: 	

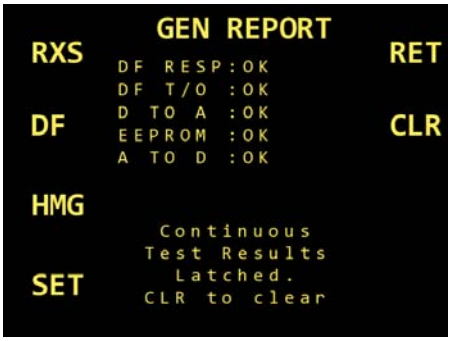
STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
		Data of beacon 2:  Data of beacon 3:  Check if the beacons are shown as a “party full” dot, “full” dot or “un-shaded” dot depending on the signal strength.	
30.	Press the SET soft-key and then press the SU DF soft-key to choose the “multi-beacon active” receiver by repeated pressing of the MBR soft-key.	The receiver may be selected as Manual, VHF, UHF or OFF.  	

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
	SETUP SCREEN		
31.	Press SET soft-key.	This screen is shown: 	
32.	Press SU DSC soft-key in SETUP screen.	This screen is shown: 	
33.	Press ENTER key in DSC ENABLES.	We can toogle between ENABLED/DISABLED: 	

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
34.	Press SU DF soft-key in SETUP screen.	This screen is shown: 	
35.	Press RIGHT/LEFT navigation key or press rotary knob.	We can change de volume level: 	
36.	Press MNT soft-key in SETUP DF screen.	We toogle between TOP/BOTTOM 	
37.	Press MBR soft-key in SETUP DF screen.	We toogle between MANUAL/VHF/UHF/OFF to choose the “multi-beacon” active receiver.	

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
38.	Press SU CSAR soft-key in SETUP screen.	This screen is shown: 	
39.	Press SU CONT soft-key in SETUP screen.	This screen is shown: 	
	MESSAGES		
40.	Press MSG soft-key in RXR STATUS screen.	This screen is shown: 	

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
41.	Press DSC soft-key in MESSAGES screen.	This screen is shown: 	
42.	Press SARSAT soft-key in MESSAGES screen.	This screen is shown: 	
	BUILT IN TEST SCREENS		
43.	Press BIT soft-key in RXR STATUS screen.	This screen is shown: 	

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
44.	Select GEN in BITE screen and press REP soft-key.	This screen is shown: 	
45.	Select DF in BITE screen and press REP soft-key.	This screen is shown: 	
46.	Select CONT in BITE screen and press REP soft-key.	This screen is shown: 	

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
47.	Select PSRI in BITE screen and press REP soft-key.	This screen is shown: 	
48.	Press TST soft-key.	No results.	

PERFORMED BY:	QUALITY ASSURANCE:	TEST MANAGER	CUSTOMER:
(Date and signature)	(Date and signature)	(Date and signature)	(Date and signature)

5.5.3. End Condition

STEP	TEST ACTION DETAILS	EXPECTED RESULTS	ACTUAL RESULTS
1.	Command MISSION REINIT	N/A	N/A