Control remoto del sintetizador de frecuencia IFR2042

L. Barbas, P. de Vicente

Informe Técnico IT-OAN 2007-6

Índice

1 Introducción

El generador de señal de bajo ruido de la serie 2040 de Aeroflex cubre el rango de 10 kHz a 5.4 GHz. Este dispositivo permite su control remoto a través del bus GPIB (General Purpose Interface Bus).

GPIB sigue el estándar IEEE 488.2 que permite enviar comandos a un instrumento, por medio del bus de datos, desde un ordenador personal o un sistema de control remoto.

En este informe se describe el sintetizador IFR2042 y el desarrollo del componente ACS y el cliente java para su control remoto.

2 Descripción del sintetizador IFR2042

El sintetizador de frecuencia IFR2042 consta de un panel frontal de cristal líquido en el que aparecen los valores que tiene configurado el dispositivo en cada instante, como la frecuencia de portadora o el nivel RF. El panel permite además visualizar cada uno de los menús en los que se configuran las opciones del dispositivo.

Fig. 1: Sintetizador IFR2042

Las principales características del sintetizador se reúnen a continuación:

Frecuencia de la portadora	• rango 10 kHZ a 5.4 GHz • resolución 0.1 Hz
Nivel RF de salida	• rango -144dBm - +13 dBm • unidades: µV, mV, V, dBm, dBv, dBmv, dBµv • resolución 0.1 dB
Incremento-decremento de valores por pasos (valores por defecto)	• Frecuencia de la portadora: 1kHz • Frecuencia LF: 1kHz • Oscilador de modulación: 1kHz • Desviación FM: 1kHz • Nivel de salida: 1dB

Modulación	• Proporciona un oscilador de modulación interna: rango de frecuencia 0,1 Hz a 500 kHz con resolución de 0,1 Hz. • Permite seleccionar un segundo oscilador de modulación • Modulación externa mediante dos entradas BNC en el panel frontal. • Máximo de 4 fuentes de modulación al mismo tiempo.
Barrido	• 4 parámetros: valores de inicio y parada, número de pasos y tiempo por paso. • No disponible con bajo ruido seleccionado.
Modos de ruido	• modo1: Bajo ruido de fase con ancho de banda AM reducido • modo2: Bajo ruido de fase con ancho de banda AM completo • Normal
Referencia de frecuencia	• Deshabilitada. • Interna 1 MHz, 5 MHz o 10 MHz • Externa 1 MHz, 5 MHz o 10 MHz
Interfaz GPIB	• Todas las funciones se pueden controlar mediante GPIB • El sintetizador puede funcionar como *talker* o *listener*.

2.1 Funcionamiento con GPIB

El generador de señal 2042 se puede manejar remotamente desde un ordenador personal mediante el bus GPIB. Todas sus funciones se pueden controlar por medio del envío de mensajes codificados a través del bus GPIB situado en la parte posterior del instrumento. El estándar implementado es el IEEE 488.2 que define los protocolos de comunicación y la sintaxis de los comandos.

El sintetizador puede funcionar como ‘talker“ o como ‘listener“. En el modo de escucha atiende los comando y solicitudes recibidos por el bus, lo que permite controlar sus funciones y establecer sus parámetros de operación. En el modo de habla permite leer su información de estado y los valores de los parámetros establecidos.

Los comandos GPBI utilizados para el control remoto del sintetizador se muestran a continuación.

PARÁMETRO	***OPERACIÓN***	***COMANDO***
Carrier Frequency	Establecer valor	`CFRQ:VALUE <valor><unidades>` `unid.: GHZ, MHZ, KHZ, HZ`
	Obtener valor	`CFRQ?`
	Establecer paso	`CFRQ:INC <valor><unidades>`
	Incrementar en un paso	`CFRQ:UP`
	Decrementar en un paso	`CFRQ:DN`

RF Level	Establecer valor	`RFLV:VALUE <valor><unidades>` `unid.:DBM, DBV, DBMV, DBUV, V, MV, UV`
	Obtener valor	`RFLV?`
	Establecer paso	`RFLV:INC <valor>`
	Incrementar en un paso	`RFLV:UP`
	Decrementar en un paso	`RFLV:DN`
	Activar salida RF	`RFLV:ON`
	Desactivar salida RF	`RFLV:OFF`
Noise Mode	Establecer modo	`IMODE <modo>` `modo: NORMAL, NOISE1, NOISE2`
Modulation	Activar	`MOD:ON`
	Desactivar	`MOD:OFF`
	Obtener estado	`MOD?`
Frequency standard	Establecer valor	`:FSTD <valor>` `valor: INT0, INT1, INT5, INT10, EXT1, EXT5, EXT10`
	Obtener valor	`FSTD?`
Error	Obtener valor	`ERROR?`

Los errores que se pueden producir al operar con el sintetizador están ordenados por números [2]. Cuando se produce un error, su número se almacena en la cola de errores y no se elimina hasta que se ejecuta el comando ‘ERROR?“, que devuelve el número de error que está en la cabecera de la cola, o se ejecuta el comando ‘*CLS“ que limpia por completo la cola de errores.

3 Componente ACS para el sintetizador

3.1 Desarrollo del componente

Todo el desarrollo del componente se ha realizado en un directorio denominado `synthesizer2042`.

El diseño del componente requiere identificar qué parámetros del sintetizador se van a definir como propiedades y qué funciones se van a definir como métodos, para establecer así la interfaz del sintetizador. El archivo `IDL` que contiene la declaración de estas propiedades y métodos, se ha denominado `synthesizer.idl` y se encuentra en el subdirectorio `idl`. En este subdirectorio también se encuentra el archivo `synthesizerError.xml` que define las excepciones lanzadas por el componente en caso de error.

El generador de código `acsGenerator` utiliza el archivo IDL para generar las plantillas en C++ a partir de las que se desarrolla el código del componente. El generador, en su forma más sencilla, se usa así:

```
> cd acsGenerator/
> acsGenerator --workdir  ~/aries21/synthesizer2042/
~/aries21/synthesizer2042/idl/synthesizer.idl
```

El generador de código crea los directorios *include, src* y *config*. Los dos primeros contienen numerosos archivos de cabecera e implementación que permiten aislar la implementación del código del interfaz del componente. Esta característica que es muy útil si se desea rehacer la interfaz (modificar el IDL) se hace a costa de multiplicar el número de ficheros. Pensamos que una vez que se ha utilizado el interfaz es mucho más claro mantener un sólo archivo para la implementación y la cabecera. De acuerdo con esta filosofía la estructura de directorios queda así:

```
|-- config
|   `-- CDB
|       `-- schemas
|           `-- synthesizer.xsd
|       `-- alma
|           `-- synthesizer.xml
|-- idl
|   |-- synthesizer.midl
|   `-- synthesizerError.xml
|-- include
|   `-- synthesizer2042Impl.h
|-- src
|   |-- Makefile
|   |-- synthesizer2042Impl.cpp
```

Si el IDL contiene propiedades ACS de tipo ENUM es necesario utilizar *macros* (`enumpropMACRO.idl`). El compilador JacORB IDL no soporta el preprocesador de macros y es necesario renombrar el archivo `syntetizer.idl` a `synthesizer.midl` (macro-idl) para poder utilizar las macros de ACS con JacORB. El Makefile genera automáticamente el fichero con extensión `.idl`.

En el subdirectorio `src` se ha creado una clase C++ que implementa la funcionalidad del sintetizador (`IFR2042.cpp`) mediante GPIB. El fichero de cabecera correspondiente se sitúa en el subdirectorio `include` (`IFR2042.h`). Los `DevIOs` se desarrollarán también en este último subdirectorio.

Por último es necesario modificar el archivo `Makefile` para que utilice los archivos adecuados y las excepciones. El `Makefile` queda de la siguiente manera:

```
# user definable C-compilation flags
USER_CFLAGS = -DCONFIG_GPIB -DNI_DRIVER

$(PROG)_OBJECTS = $(PROG)
$(PROG)_LDFLAGS =
$(PROG)_LIBS = C++

#
# Includes (.h) files (public only)
# ---------------------------------
INCLUDES = IFR2042.h synthesizer2042Impl.h
```

```
#
# Libraries (public and local)
# ---------------------------
LIBRARIES = IFR2042 synthesizer2042

#
# <brief description of IFR2042 library>
IFR2042_OBJECTS =  IFR2042
IFR2042_LDFLAGS =
IFR2042_LIBS    =  synthesizerError

#
# <brief description of synthesizer2042 library>
synthesizer2042_OBJECTS =  synthesizer2042Impl
synthesizer2042_LDFLAGS =
synthesizer2042_LIBS=IFR2042 synthesizerStubs synthesizerError

...

#
# Configuration Database Files
# ---------------------------
CDB_SCHEMAS = synthesizer


# IDL Files and flags
#
IDL_FILES = synthesizer
IDL_TAO_FLAGS =
USER_IDL =

ACSERRDEF = synthesizerError
```

3.2 Descripción del componente. Clase C++

Se ha desarrollado una clase C++ (`IFR2042`) que implementa la funcionalidad del sintetizador por medio de GPIB. A continuación se describen las variables, funciones públicas y funciones privadas utilizadas.

Se han definido enums para los parámetros del sintetizador que aceptan varias unidades: la frecuencia de portadora y el nivel de RF, así como para los parámetros que pueden tomar varios valores: el ruido y la frecuencia de referencia.

```
enum unitFreq {freqGHZ, freqMHZ, freqKHZ, freqHZ};
enum unitLevel {DBM, DBV, DBMV, DBUV, V, MV, UV};
enum noiseMode {NORMAL, NOISE1, NOISE2};
enum freqStandard {INT0, INT1, INT5, INT10, EXT1, EXT5, EXT10};
```

El generador de señales permite seleccionar el nivel de RF del sintetizador, las unidades, el paso, así como activar o desactivar su salida. Toda esta información se guarda en una estructura del siguiente tipo:

```
typedef struct dataRFLV
{
      double level;
      unitLevel unit;
      bool status; // True=ON false=OFF(default)
      double step;
}dataRFLV;
```

La frecuencia de portadora y el paso con el que se cambia se almacenan en una estructura de este tipo:

```
typedef struct dataCFRQ
{
      double frequency;
      double step;
}dataCFRQ;
```

Las unidades son siempre Hz.

Funciones públicas:

- El **constructor** establece el índice de la tarjeta GPIB y las direcciones primaria y secundaria del dispositivo. Crea el descriptor del dispositivo para poder enviar los comandos a través de este descriptor. Por último limpia la cola de mensajes de error que pudiera tener el sintetizador en el momento de iniciar su control remoto.

  ```
  IFR2042(const int & boardID, const int & primaryAddress, const
  int & secondaryAddress) throw (synthesizerError::deviceErrorExImpl&,
  synthesizerError::writeErrorExImpl &,
  synthesizerError::readErrorExImpl &);
  ```

  Parámetros:
  - `boardID`: índice de la tarjeta(GPIB0=0,GPIB1=1,etc)
  - `primaryAddress`: dirección primaria del dispositivo.
  - `secondaryAddress`: dirección secundaria del dispositivo.

- El **destructor** deja el dispositivo fuera de línea.

  ```
  ~IFR2042() throw (synthesizerError::deviceErrorExImpl &);
  ```

- La función **setFrequency** permite establecer la frecuencia de la portadora en GHz, MHz, KHz o Hz. Comprueba si el valor está dentro del rango permitido (10kHz - 5.4GHz). En caso afirmativo envía el comando correspondiente, de lo contrario ajusta el valor al máximo o al mínimo permitido según corresponda y envía el comando.

  ```
  void setFrequency(const double & freq, const unitFreq & unit)
  throw (synthesizerError::writeErrorExImpl &);
  ```

  Parámetros:
  - `freq`: valor de frecuencia
  - `unit`: enum de unidades

- La función **setStepFrequency** establece el paso de la frecuencia de portadora en Hz. Comprueba si el valor está dentro del rango permitido (10kHz - 5.4GHz). En caso afirmativo envía el comando correspondiente, de lo contrario ajusta el valor al máximo o al mínimo permitido según corresponda y envía el comando.

```
void setStepFrequency(const double & step ) throw
(synthesizerError::writeErrorExImpl &);
```

Parámetros:
- `freq`: paso de frecuencia.

- La función **upFrequency** incrementa la frecuencia un paso.

```
void upFrequency() throw (synthesizerError::writeErrorExImpl &);
```

- La función **downFrequency** decrementa la frecuencia un paso.

```
void downFrequency() throw(synthesizerError::writeErrorExImpl &);
```

- La función **getFrequency** obtiene la frecuencia de la portadora en Hz y el paso establecido. La función formatea una cadena del tipo:
  `:CFRQ:VALUE frecuency_value;INC frequency_step`
  procedente del sintetizador y almacena los valores en la estructura `dataCFRQ`.

```
const dataCFRQ getFrequency() throw
(synthesizerError::writeErrorExImpl &,
synthesizerError::readErrorExImpl &);
```

- La función **setRFLevel** permite establecer el nivel RF en dBm, dBV, dBmV, dBuV, V, mV o uV. Comprueba si el valor está dentro del rango permitido (-144 dBm a 13 dBm). En caso afirmativo envía el comando correspondiente, de lo contrario ajusta el valor al máximo o al mínimo permitido según corresponda y envía el comando.

```
void setRFLevel(const double & level, const unitLevel & unit)
throw (synthesizerError::writeErrorExImpl &);
```

Parámetros:
- `level`: valor del nivel RF
- `unit`: enum de unidades

- La función **setStepRFLevel** establece el paso del nivel de RF, únicamente en DB. Comprueba si el valor está dentro del rango permitido (-144 dBm a 13 dBm). En caso afirmativo envía el comando correspondiente, de lo contrario ajusta el valor al máximo o al mínimo permitido según corresponda y envía el comando.

```
void setStepRFLV(const double & step) throw
(synthesizerError::writeErrorExImpl &);
```

Parámetros:
- `freq`: valor del paso del nivel RF.

- La función **upRFLV** incrementa el nivel RF un paso.

```
void upRFLV() throw (synthesizerError::writeErrorExImpl &);
```

- La función **downRFLV** decrementa el nivel RF un paso.

```
void downRFLV() throw (synthesizerError::writeErrorExImpl &);
```

- La función **dataRFLV** obtiene el nivel RF, las unidades, el estado y el paso utilizado. La función formatea una cadena del tipo:
`:RFLV:UNITS <units>;VALUE <value>;INC frequency_step;level_status`
procedente del sintetizador y almacena los valores en la estructura `dataRFLV` .

```
const dataRFLV getRFLevel() throw
(synthesizerError::writeErrorExImpl &,
synthesizerError::readErrorExImpl &);
```

- La función **setStatusRFLV** activa o desactiva la salida RF.

```
void setStatusRFLV (const bool & status) throw
(synthesizerError::writeErrorExImpl &);
```

Parámetro:
- `status`: estado ON=activar OFF=desactivar.

- La función **setNoiseMode** establece el modo de ruido a NOISE1, NOISE2 o NORMAL.

```
void setNoiseMode (const noiseMode & mod) throw
(synthesizerError::writeErrorExImpl &);
```

Parámetro:
- `mod`: Modo

- La función **setModulationStatus** activa o desactiva la modulación.

```
void setModulationStatus (const bool & status) throw
(synthesizerError::writeErrorExImpl &);
```

Parámetro:
- `status`: estado ON=activar OFF=desactivar.

- La función **getModulationStatus** permite obtener el estado de la modulación. La función formatea una cadena del tipo: `:MOD:modulation_status` procedente del sintetizador y almacena en una variable `bool` un valor `true` si está activada y un valor `false` en caso contrario.

```
const bool getModulationStatus () throw
(synthesizerError::writeErrorExImpl &,
synthesizerError::readErrorExImpl &);
```

- La función **setFrequencyStandard** establece la frecuencia de referencia según las siguientes opciones:
  -Desactivada
  -Interna 1 MHz, 5MHz o 10MHz
  -Externa 1 MHz, 5MHz o 10MHz.

```
void setFrequencyStandard (const freqStandard & freqSTD)    throw
(synthesizerError::writeErrorExImpl &);
```

Parámetro:
- `freqSTD`: enum de opciones de frecuencia de referencia.

- La función **freqStandard** permite obtener la frecuencia de referencia utilizada. La función formatea una cadena del tipo: `:FSTD frequency_standard` procedente del sintetizador y devuelve el modo establecido mediante un enum `freqStandard.` En este caso la respuesta del dispositivo termina con un "\n"

```
const freqStandard getFrequencyStandard () throw
(synthesizerError::writeErrorExImpl &,
synthesizerError::readErrorExImpl &);
```

- La función **getError** obtiene el último mensaje de error ocurrido y almacena en una cadena de texto la descripción correspondiente.

```
char * getError () throw     (synthesizerError::writeErrorExImpl &,
synthesizerError::readErrorExImpl &);
```

El envío de instrucciones al sintetizador se hace utilizando en todos los casos la rutina:

ibwrt(Device,cmd,strlen(cmd))

Parámetros:
- `Device`: descriptor del dispositivo.
- `cmd`: cadena de caracteres que contiene el comando.
- `strlen(cmd)`: longitud del comando.

La respuesta del sintetizador se lee empleando la rutina:

ibrd(Device, response, length)

Parámetros:
- `Device`: descriptor del dispositivo.
- `response`: cadena de caracteres que almacena la respuesta.
- `length`: longitud de la respuesta.

Funciones privadas:

- La función **cleanErrorQueue** vacía la cola de errores del sintetizador.

```
void cleanErrorQueue() throw  (synthesizerError::writeErrorExImpl &,
synthesizerError::readErrorExImpl &);
```

Variables privadas:

- `m_boardID`: Índice de la tarjeta GPIB
- `m_primaryAddress`: dirección primaria del dispositivo
- `m_secondaryAddress`: dirección secundaria del dispositivo
- `Device`: descriptor
- `m_msg`: cadena de caracteres que almacena el mensaje de error

3.3 Errores

Las excepciones que lanza el componente ACS del sintetizador cuando se produce un error están definidas en el archivo `synthesizerError.xml`. El número que agrupa estas excepciones es el 2006.

Se han definido 4 tipos de excepciones:
- `deviceError:` se lanza si el dispositivo no se puede inicializar, limpiar o cerrar mediante GBIB.
- `writeError:` se lanza si se produce un error al enviar comandos al sintetizador.
- `readError:` se lanza si se produce un error al recibir datos del sintetizador.
- `paramOutOfLimits:` se lanza si los valores de los parámetros del sintetizador se salen del rango permitido.

3.4 IDL

Como se ha comentado anteriormente el archivo IDL define la interfaz del sintetizador. Se ha creado un modulo denominado `synthesizer2042`. Este módulo contiene la definición de la interfaz `synthesizer`, que hereda de `ACS::CharacteristicComponent`, en la que se definen las propiedades y métodos que permiten utilizar el sintetizador.

En el módulo se definen varios `enums` que describen los valores que pueden tomar ciertos parámetros del sintetizador y que se utilizan para definir propiedades. También se definen dos estructuras que permiten almacenar los valores retornados por los métodos del componente.

```
enum suffixCFRQ {freqGHZ, freqMHZ, freqKHZ, freqHZ};
ACS_ENUM(suffixCFRQ);

enum suffixRFLV {rfDBM, rfDBV, rfDBMV, rfDBUV, rfV,
        rfMV, rfUV};
ACS_ENUM(suffixRFLV);

enum suffixStatusRFLV {rfON, rfOFF};
ACS_ENUM(suffixStatusRFLV);

enum suffixIMODE {NORMAL, NOISE1, NOISE2};
ACS_ENUM(suffixIMODE);

enum suffixFSTD {INT0, INT1, INT5, INT10, EXT1, EXT5,
          EXT10};
ACS_ENUM(suffixFSTD);

enum suffixModulation {modON, modOFF};
ACS_ENUM(suffixModulation);

struct RFLevelData
{
        double value;
        suffixRFLV unit;
        boolean status;
        double step;
};

struct CarrierFrequencyData
{
        double value;
        double step;
};
```

3.4.1 Propiedades

- `ACS::RWdouble stepFrequency:` paso utilizado para la frecuencia de portadora.
- `ACS::RWdouble stepRFLevel:` paso utilizado para el nivel RF.
- `RWsuffixStatusRFLV statusRFLevel:` Estado en el que se encuentra el nivel RF: activado o desactivado.
- `RWsuffixIMODE instrumentMode:` Modo de ruido del sintetizador.
- `RWsuffixFSTD frequencyStandard:` Frecuencia de referencia del sintetizador.

- `RWsuffixModulation modulation:` estado de la modulación: activado o desactivado.
- `ACS::ROstring error:` (sólo lectura) Error producido al operar con el sintetizador.

3.4.2 Métodos

Se han definido los siguientes métodos:

- Para establecer y obtener la frecuencia de la portadora:

```
void setCarrierFrequency(in double value, in suffixCFRQ unit)
                         raises (synthesizerError::writeErrorEx);
CarrierFrequencyData getCarrierFrequency() raises
                         (synthesizerError::writeErrorEx,
                          synthesizerError::readErrorEx);
```

- Para incrementar y decrementar la frecuencia de portadora en paso.

```
void upCarrierFrequency() raises (synthesizerError::writeErrorEx);
void downCarrierFrequency() raises(synthesizerError::writeErrorEx);
```

- Para establecer y obtener el nivel RF:

```
void setRFLevel(in double value, in suffixRFLV unit) raises
                                 (synthesizerError::writeErrorEx);
RFLevelData getRFLevel() raises(synthesizerError::writeErrorEx,
                                  synthesizerError::readErrorEx);
```

- Para incrementar y decrementar el nivel RF un paso.

```
void upRFLevel() raises (synthesizerError::writeErrorEx);
void downRFLevel() raises (synthesizerError::writeErrorEx);
```

3.5 Implementación del componente

Para implementar el componente se utilizan los ficheros `synthesizer2042Impl.cpp`, `synthesizer2042Impl.h` y los DevIOs.

Para cada propiedad se crea un DevIO<tipo_propiedad>, que hereda de la clase DevIO, y se le pasa el tipo de la propiedad correspondiente:

```
DevIOerror: public DevIO<ACE_CString>
DevIOfrequencyStandard: public DevIO<synthesizer2042::suffixFSTD>
DevIOinstrumentMode: public DevIO<synthesizer2042::suffixIMODE>
DevIOmodulation: public DevIO<synthesizer2042::suffixModulation>
DevIOstatusRFLevel: public DevIO<synthesizer2042::suffixStatusRFLV>
DevIOstepFrequency: public DevIO<CORBA::Double>
DevIOstepRFLevel: public DevIO<CORBA::Double>
```

Los DevIOs toman como referencia un objeto de la clase `IFR2042` de modo que tienen acceso a todas sus funciones públicas descritas en la sección 3.2. En cada uno de los DevIOs se implementan métodos de lectura (`read ()`) y escritura (`write()`) que llaman a las funciones correspondientes de la clase `IFR2042`.

El método `initialize()` de la clase `synthesizerImpl` instancia los DevIOs y crea las propiedades. Las propiedades obtienen su valor predeterminado tras consultar la base de datos (`synthesizer.xml`) empleando el método `write()` de los DevIOs.

La implementación de los métodos definidos en el IDL también se realiza en la clase `synthesizerImpl`. En el método `initialize()` se crea un objeto de la clase `IFR2042` para tener acceso a todas sus funciones públicas. Cada método realiza una llamada a la función de la clase C++ `IFR2042` correspondiente. En la mayoría de los métodos es necesario hacer una conversión entre los enums definidos en el IDL y los definidos en la clase `IFR2042`.

3.6 Archivo CDB

El archivo `synthesizer.xml` situado en el subdirectorio *config* contiene los valores predeterminados que toman las propiedades en el momento de su creación.

Por ejemplo, el contenido del fichero para la propiedad *stepRFLevel* es:

```
<stepRFLevel
    units="dB"
    default_value="1.0"
    min_value="0.0"
    max_value="157.0"
    min_step="0.1"
    default_timer_trig="2.0"
    min_timer_trig="1"
    min_delta_trig="1"
/>
```

4 Cliente Java para el sintetizador

Se ha desarrollado un cliente Java para el control remoto del sintetizador IFR2042. El cliente implementa una interfaz gráfica que permite interactuar con el sintetizador de un modo sencillo.

4.1 Diseño detallado

El cliente se implementa con dos clases. La clase `synthesizerClient` interactúa con el componente ACS del sintetizador y la clase `synthesizerGUI` crea y gestiona la interfaz gráfica del cliente.

4.1.1 Diagrama de clases

La clase `synthesizerClient` contiene el método `main()`. Esta clase crea el cliente ACS para el sintetizador y permite la visualización de la interfaz gráfica.

La clase `synthesizerClient` hereda de `alma.acs.component.client.ComponentClient`. La comunicación entre el cliente y el componente se realiza a través de sus propiedades y métodos.

La interfaz gráfica, que muestra las opciones de configuración del dispositivo, está implementada en la clase `synthesizerGUI` que hereda de `javax.swing.JFrame`.

La figura siguiente muestra el diagrama de clases del cliente; aparecen las clases diseñadas y su relación. Las flechas indican una relación de dependencia, el origen de la flecha indica la clase de la que depende la que está en el extremo de la flecha.

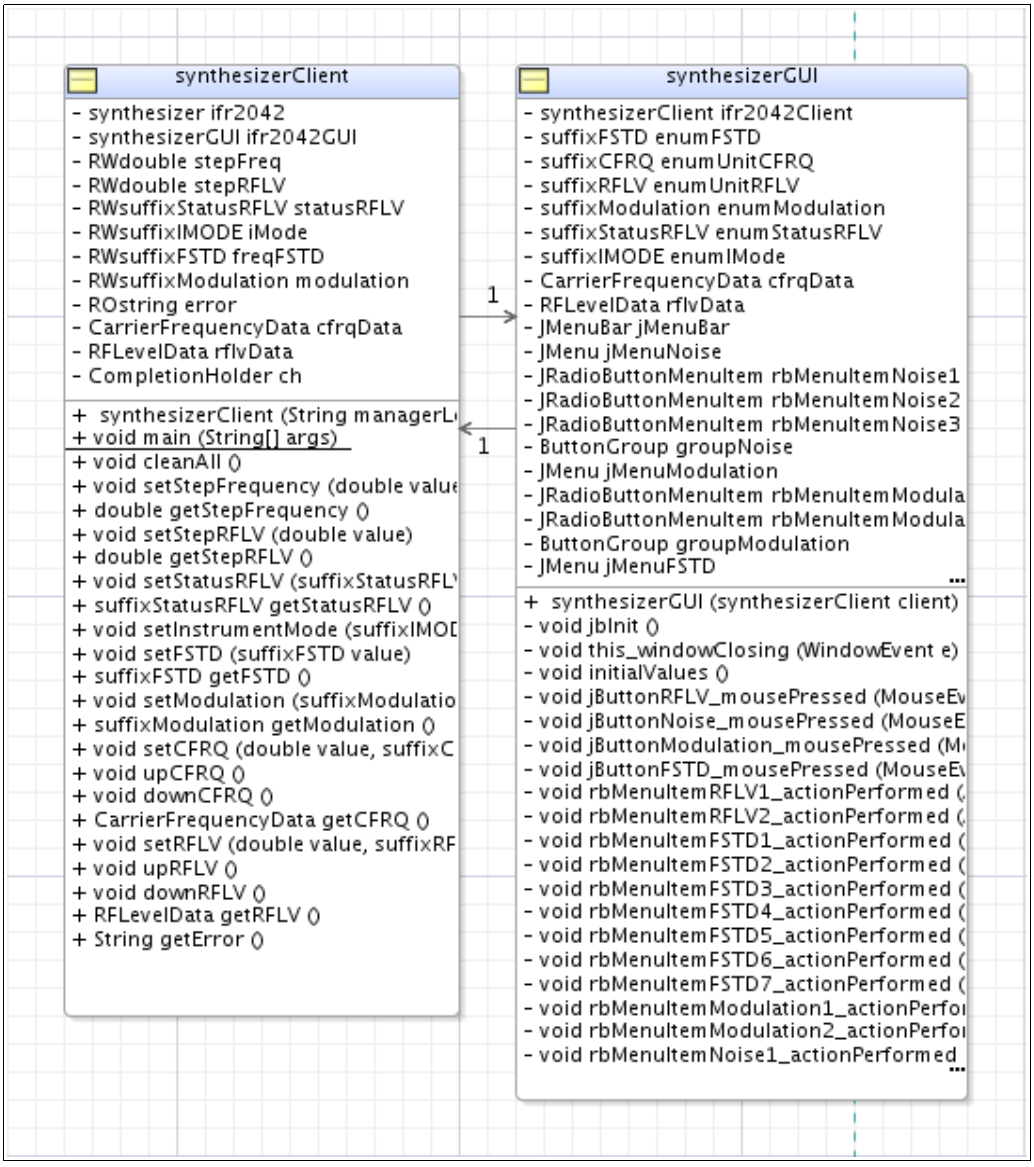


Fig. 2. Diagrama general de clases

Ambas clases forman parte del paquete `alma.oansynthesizer2042`. Los archivos fuente `.java` se deben situar en un directorio denominado `alma/oansynthesizer2042`. La compilación de estos ficheros y la generación del paquete `synthesizerClient.jar`

que incluye ambas clases se consigue añadiendo las siguientes líneas en el archivo `Makefile`:

```
#
# Jarfiles and their directories
#
JARFILES= synthesizerClient
synthesizerClient_DIRS= alma
synthesizerClient_EXTRAS= alma/oansynthesizer2042/images/down.png
                          alma/oansynthesizer2042/images/up.png
```

El directorio `alma` debe estar en el mismo directorio que el archivo `Makefile`.

4.1.2 Paquete alma.oansynthesizer2042

En esta sección se describen en detalle cada una de las clases del paquete `alma.oansynthesizer2042`.

4.1.2.1 La clase synthesizerClient

Contiene el método `main()` que permite iniciar la aplicación. El cliente se inicia ejecutando el siguiente comando:

```
acsStartJava alma.oansynthesizer2042.synthesizerClient
```

El argumento pasado a `acsStartJava` es el nombre de la clase que contiene el método `main()` precedido del paquete al que pertenece.

En el constructor de esta clase, llamado desde `main()`, se crea un objeto del componente 'synthesizer" y se obtiene una referencia a cada una de las propiedades del componente. Para que los valores de estas propiedades se puedan obtener y establecer desde cualquier otra clase del cliente, se han definido métodos públicos `getXXX` y `setXXX` para cada propiedad. Del mismo modo se han implementado funciones públicas que permiten el uso de los métodos del componente ACS del sintetizador.

En el constructor también se crea un objeto de la interfaz gráfica y se permite su visualización.

4.1.2.2 La clase synthesizerGUI

La clase `synthesizerGUI` implementa la interfaz gráfica que permite el control remoto del sintetizador. En la sección 4.3 se muestra el formato completo de la interfaz.

Esta clase hereda de `javax.swing.JFrame` y, por tanto, muestra una ventana de alto nivel con título y borde (*frame*).

El constructor de la clase recibe como parámetro un objeto de la clase `synthesizerClient`, para poder tener acceso completo a todos sus métodos. También se realiza una llamada a los métodos `jbInit()` e `initialValues()`.

El método `jbInit()` contiene el código que desarrolla el formato de la ventana gráfica: etiquetas, cajas de texto, listas desplegables, botones, etc.

La mayoría de los elementos de la interfaz implementan la interface `java.awt.event.ActionListener` de modo que están a la 'escucha" de cualquier modificación que el usuario realice sobre ellos. Cuando esto ocurra, se llamará al método `XXX_actionPerformed()` encargado de realizar las funciones correspondientes, como por ejemplo, modificar el valor de una propiedad a través del objeto *synthesizerClient*.

El método `initialValues()` obtiene las opciones configuradas en el sintetizador en el momento del inicio de la aplicación y los muestra en la interfaz gráfica.

4.2 Detalles de Implementación

4.2.1 Monitorización

El sintetizador se puede controlar remotamente a través de GPIB o manualmente desde su panel frontal. La operación remota simultánea del sintetizador no es posible. En el código del componente se han definido `muttex` que impiden la escritura simultánea de dos procesos. Sin embargo es posible que un usuario modifique los valores del sintetizador inmediatamente después que otro. Para evitar que la interfaz gráfica muestre información falsa u obsoleta es conveniente leer periódicamente la configuración del sintetizador.

La clase `javax.swing.Timer` obtiene la configuración del sintetizador y la muestra cada 2 minutos. En la clase `synthesizerGUI` se crea un objeto de esta clase.

A continuación aparece la parte de código correspondiente a la monitorización de variables del sintetizador como la referencia de frecuencia, estado de la modulación, etc.:

```
Timer timer2;
  ...
//Modulation, instrument mode Frequency Standard,...
timer2 = new Timer(120000, new ActionListener ()
{
    public void actionPerformed(ActionEvent e)
    {
        initialValues();
    }
});
timer2.start();
```

Cada 2 minutos se ejecutará el método `initialValues()` que obtiene la configuración del sintetizador y la muestra en la interfaz gráfica.

Antes de terminar la aplicación será necesario detener el `timer` mediante la función `timer2.stop()`.

Todos los valores del sintetizador se monitorizan cada 2 minutos, salvo la frecuencia de portadora y el nivel RF que se monitorizan cada 2 segundos.

4.2.2 Mensajes de error del sintetizador

Como vimos en la sección 2.1 el sintetizador almacena el código de los errores producidos en una cola de error. La propiedad *error* del componente permite obtener la descripción del último error producido.

En el cliente, después de realizar una acción que pueda producir un error, se ejecuta la función `showError()`.

```
private void showError()
{
      String e = ifr2042Client.getError();
      if(e.compareTo("Error queue EMPTY") != 0)
      {
          jLabelError.setText("Error");
          jLabelError.setVisible(true);
          jButtonShowError.setVisible(true);
          this.setErrorLog(e);
      }
 }
```

Esta función obtiene el valor de la propiedad *error*. Si no se ha producido error se recibe la descripción "Error queue EMPTY". En caso contrario se realiza una llamada a `setErrorLog()`que almacena la descripción del error producido en un área de texto con la hora y fecha de ocurrencia.

```
 private void setErrorLog(String val)
{
  String str = val;
  Calendar cal = Calendar.getInstance();
  Date fecha = cal.getTime();
  SimpleDateFormat formato = new SimpleDateFormat
                            ("yyyy.MM.dd 'at' hh:mm:ss");
  logError = logError.concat("\n");
  logError = logError.concat(" ");
  logError = logError.concat(formato.format(fecha));
  logError = logError.concat("-");
  logError = logError.concat(val);
  logErrorTextArea.setText(logError);

}
```

El usuario pueda visualizar este área de texto en cualquier momento pulsando un botón, que únicamente aparecerá cuando se produzca error (Ver `jButtonShowError.setVisible(true)` en la función `showError()`).

4.3 Interfaces gráficas

En esta sección se describe el cliente gráfico del sintetizador IFR2042. Al iniciar la aplicación se muestran en pantalla los valores que el sintetizador tiene configurados en ese momento. Estos valores corresponden a:

- Frecuencia de la portadora: valor + unidades
- Nivel de RF: valor + unidades + estado
- Frecuencia de referencia: referencia
- Modo de ruido: Modo
- Modulación: estado

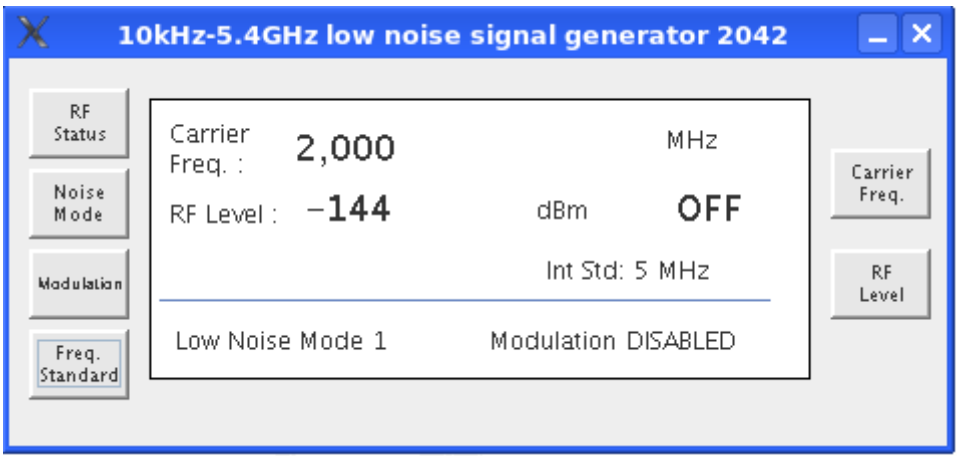


Fig. 3. Interfaz gráfica principal

A ambos lados del panel central aparecen una serie de botones que permiten variar la configuración del dispositivo por medio de menús.

4.3.1 Botón RF Status

Este botón, situado en la parte izquierda de la interfaz, muestra un menú desplegable con las posibilidades de configuración del *estado del nivel RF* (ON/OF). Cuando se selecciona una de las opciones el valor en la pantalla principal cambia.

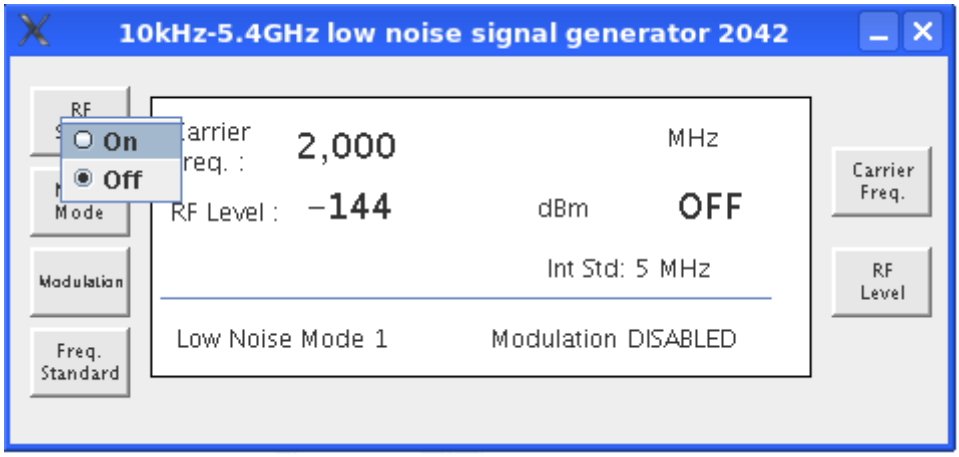


Fig. 4. Menú desplegable del botón 'R F Status"

4.3.2 Botón Noise Mode

Este botón, situado en la parte izquierda de la interfaz, muestra un menú desplegable con las posibilidades de configuración del *modo de ruido* (NORMAL,

NOISE1, NOISE2). Cuando se selecciona una de las opciones el valor en la pantalla principal cambia.

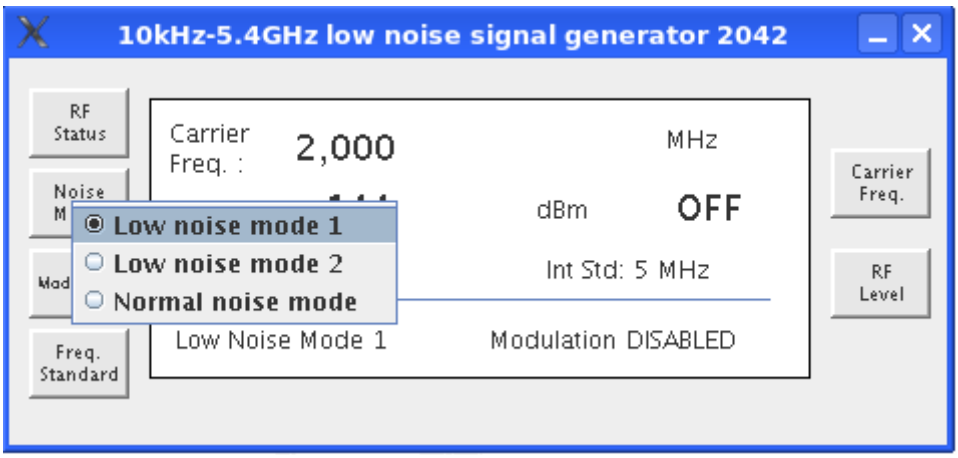


Fig. 5. Menú desplegable del botón 'Noise Mode"

4.3.3 Botón Modulation

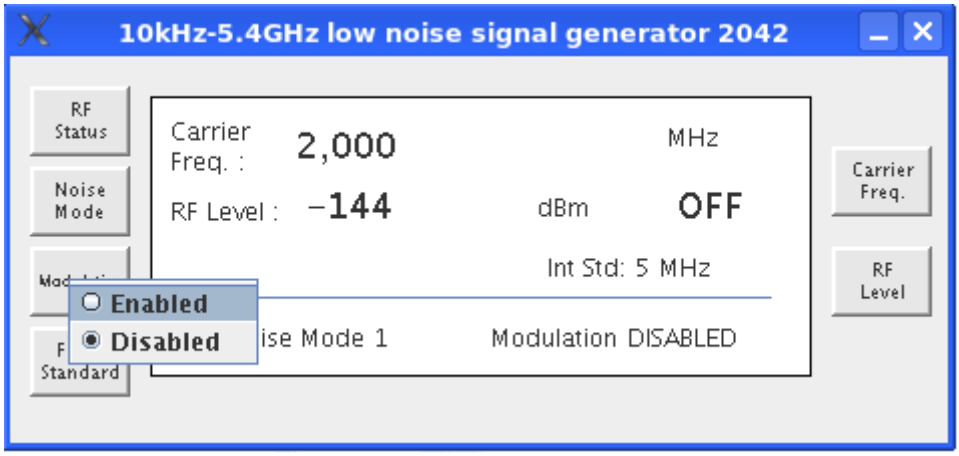


Fig. 6. Menú desplegable del botón ' Modulation"

Este botón, situado en la parte izquierda de la interfaz, muestra un menú desplegable con las posibilidades de configuración del *estado de la modulación* (ENABLED/DISABLED). Cuando se selecciona una de las opciones el valor en la pantalla principal cambia.

4.3.4 Botón Frequency Standard

Este botón, situado en la parte izquierda de la interfaz, muestra un menú desplegable con las opciones de configuración de la *frecuencia de referencia* (INT0, INT1, INT5, INT10, EXT1, EXT5, EXT10). Cuando se selecciona una de las opciones el valor en la pantalla principal cambia.

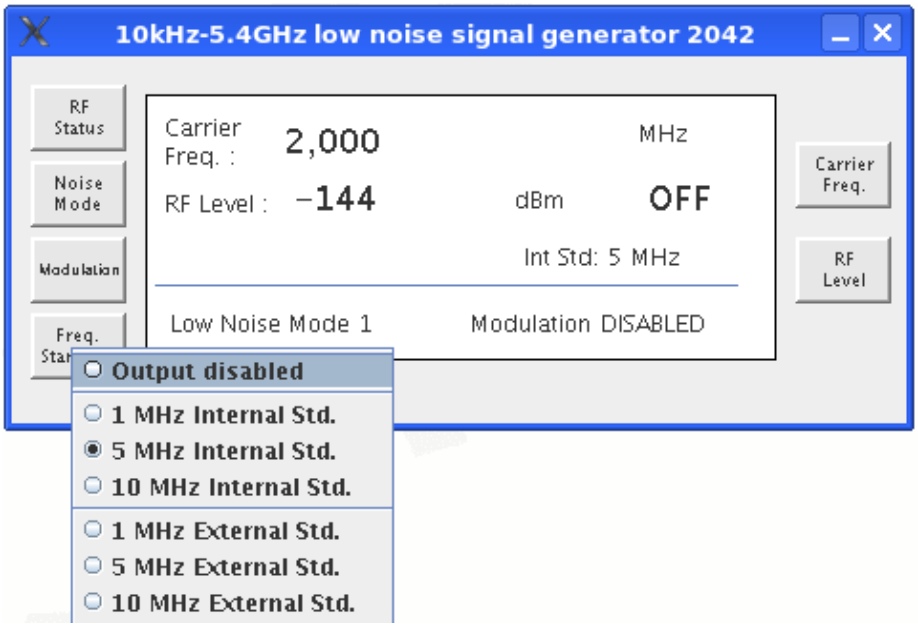


Fig. 7. Menú desplegable del botón 'Freq. Standard"

4.3.5 Menú Carrier Frequency

Este botón, situado en la parte derecha de la interfaz, muestra en la ventana central las opciones de configuración de la *frecuencia de portadora*.

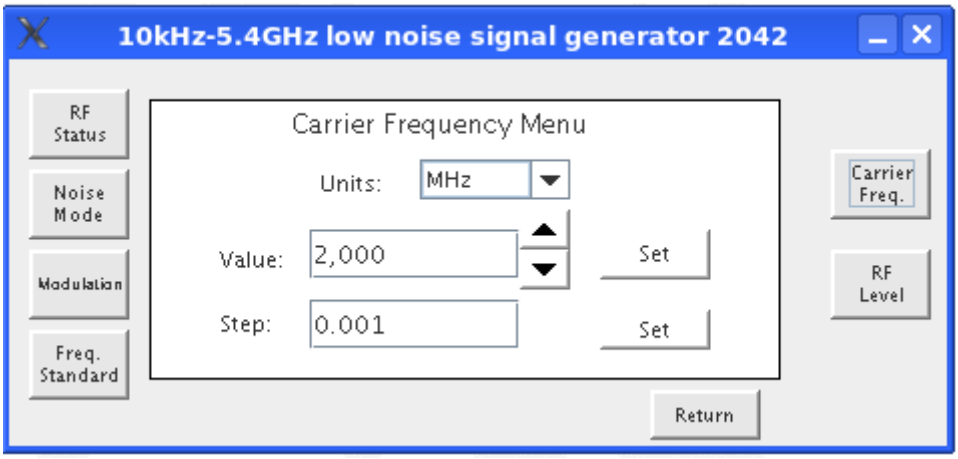
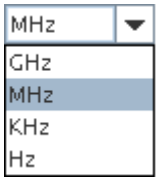


Fig. 8. Menú del botón ' Carrier Freq."

En este menú se pueden seleccionar:

- Unidades



- Valor
- Paso

Para establecer el valor de frecuencia o el paso es necesario pulsar el botón ‘Set“. Una vez establecidas las opciones deseadas hay que pulsar el botón ‘Return“ para volver a la pantalla principal.

4.3.6 Menú RF Level

Este botón, situado en la parte derecha de la interfaz, muestra en la ventana central las opciones de configuración del *nivel RF*.

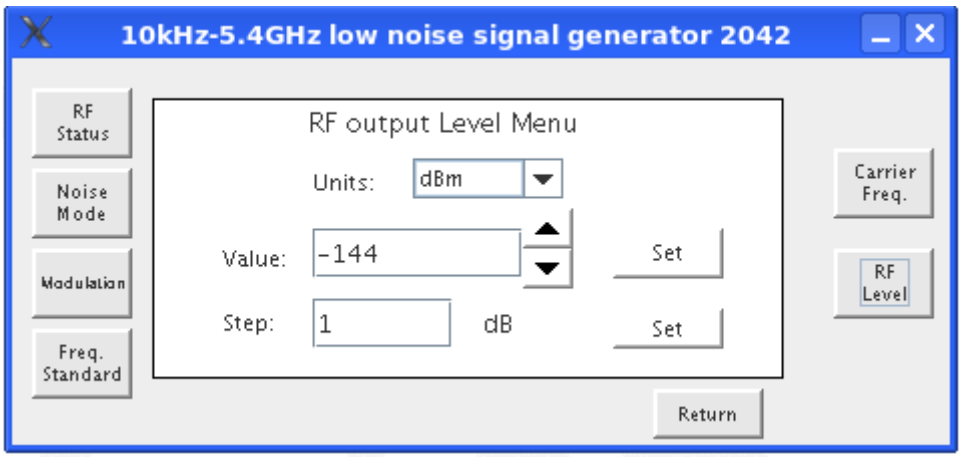
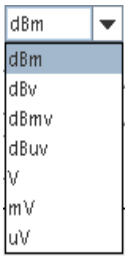


Fig. 9. Menú del botón ‘RF Level“

En este menú se pueden seleccionar:

- Unidades



- Valor
- Paso

Para establecer el valor del nivel RF o el paso es necesario pulsar el botón ‘Set“. Una vez establecidas las opciones deseadas hay que pulsar el botón ‘Return“ para volver a la pantalla principal.

4.3.7 Errores

Si en el sintetizador se produce un error, en la interfaz gráfica aparece una etiqueta en color rojo denominada “Error” y junto a ésta un botón denominado “Show Details”.

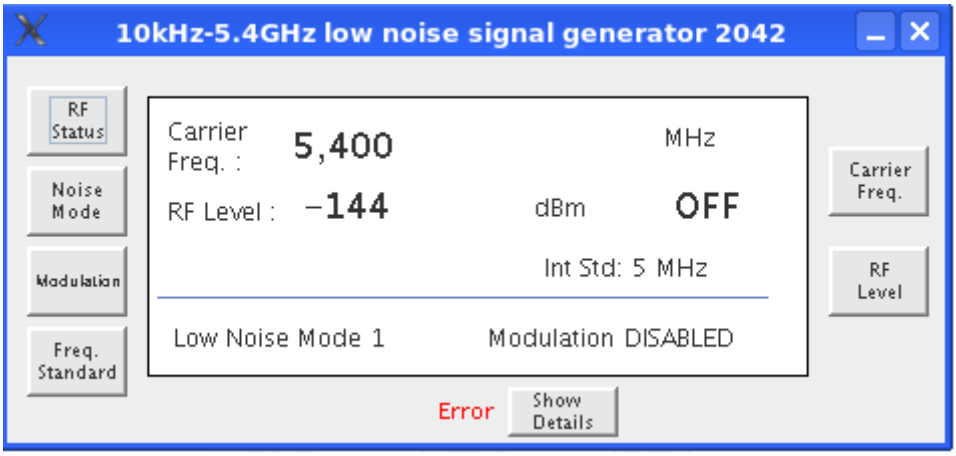


Fig. 10. Botón 'Sho w Details"

Al pulsar el botón "Show Details" se muestran en la ventana central los errores ocurridos y el instante de ocurrencia. Una vez que el usuario ha visualizado los errores, el botón de error y la etiqueta desaparecen de la interfaz.

Si el botón Error no se pulsa, permanecerá en la interfaz hasta que el usuario visualice los errores. Si se producen más errores estos se añaden a los anteriores.

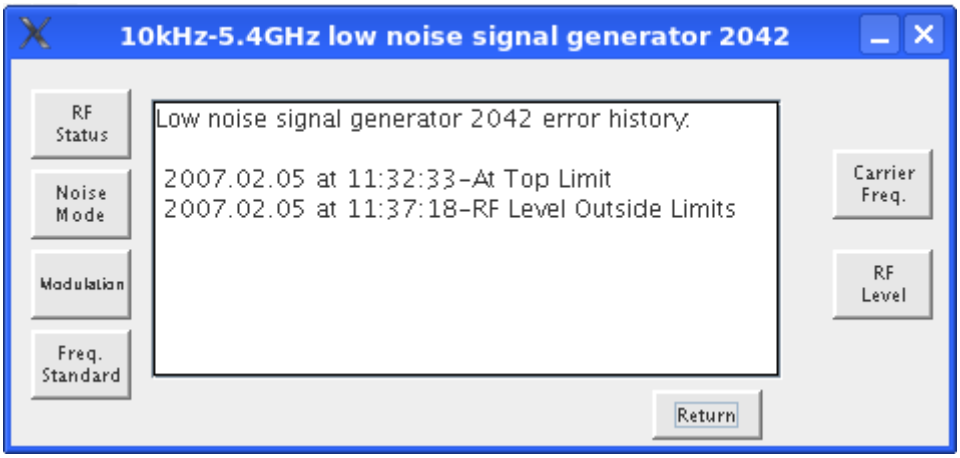


Fig. 11. Historial de errores

5 Referencias

[1] FROUFE, A. *JAVA 2: Manual de usuario y tutorial, 2ª Ed*. Madrid: Edit. RA-MA, 2000.

[2] AEROFLEX. *AM/FM Signal Generators 2040 Series operating manual*. Mayo 2005.

[3] NATIONAL INSTRUMENTS. *NI-488.2 Software Reference Manual for MS-DOS*. Mayo1992.

[4] JAVA Homepage: http://java.sun.com/

[5] ACS Homepage: http://www.eso.org/~gchiozzi/AlmaAcs/