
C-4
Villaverde Bajo
Atocha

INTRODUCCIÓN

El objetivo de este documento es detallar el funcionamiento de las aplicaciones software desarrolladas por Textel Marimón S.A. para los teleindicadores de andén de Renfe.

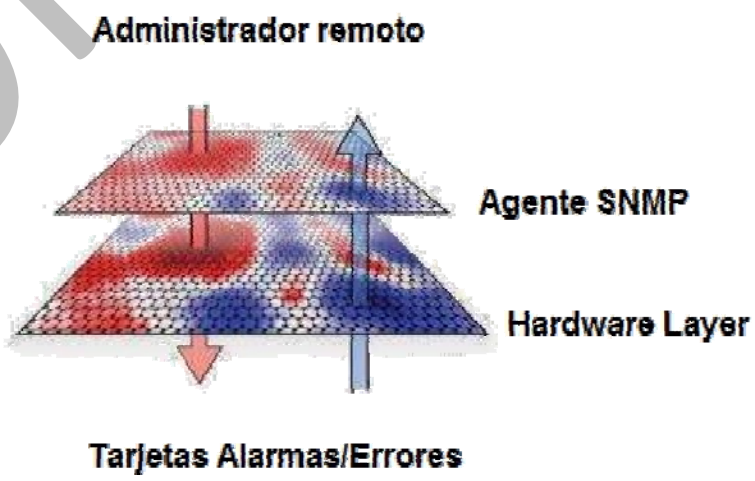
Los teleindicadores se encargan de mostrar al viajero cualquier contenido generado por el Administrador. Normalmente serán animaciones para avisar del destino y paradas intermedias del próximo tren. Un ordenador industrial en el interior del teleindicador, preparado para trabajar en rangos de temperatura extendida, se encargará de las tareas de reproducción y control del hardware.

Se suministran dos aplicaciones: una desarrollada en C++, que implementa un **watchdog** que vigila la consistencia del sistema operativo, y otra **aplicación principal,** desarrollada en Java, para el resto de tareas (monitorización de sensores y agente SNMP).

La **aplicación principal** trabaja en dos capas:

- La capa superior, **AGENTE SNMP**, encargada de responder con los datos del árbol las peticiones recibidas desde el Administrador de la red para conocer el estado de cualquier componente del hardware, así como de lanzar las notificaciones (alarmas) que se generen.

- La capa inferior, **CAPA HARDWARE**, encargada de pedir cíclicamente las lecturas de los sensores y mantener actualizado el árbol de datos, y responsable de lanzar a la capa superior las alarmas detectadas.



CONFIGURACIÓN

Al iniciarse la aplicación principal, la primera tarea es leer los archivos de configuración:

- **address.xml** (con los datos particulares del display)
- **config.xml** (con la definición de variables)
- **TEXTEL-TLIA.mib** (con la estructura de datos del agente snmp)

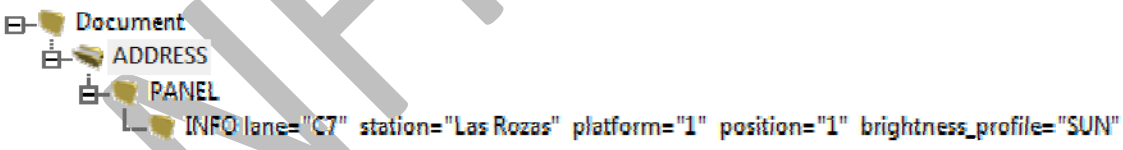
1. ADDRESS.XML

Este archivo 'address.xml' es distinto para cada uno de los paneles de la red.

Es el archivo que identifica inequívocamente un panel…

- línea
- estación
- andén
- posición en el andén

Con la aplicación se suministra un editor de archivos XML, que permitirá ver este archivo de una forma parecida a ésta:



… y que se debería interpretar de la siguiente forma:

es el primer teleindicador, más cercano a Atocha (position="1"),

del andén 1 (platform="1"),

de la Estación de Las Rozas,

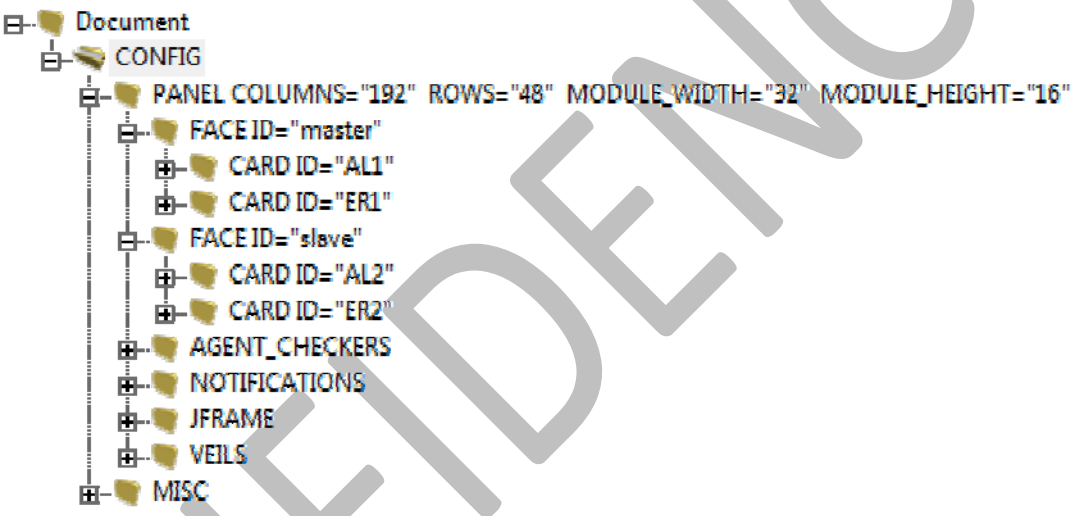
en la línea C7.

Está instalado en una ubicación con abundante luz natural.

2. CONFIG.XML

Para un proyecto concreto, este archivo 'config.xml' será seguramente igual en todos los paneles de ese mismo tamaño.

Es el archivo en el que se configura el comportamiento del panel, y hemos separado esta configuración "general" de la de los datos particulares del display (descritos en el punto anterior, distintos en cada panel) para que, ante cualquier cambio de criterio en la configuración de una serie de paneles, el administrador solamente tenga que copiar el mismo archivo "config.xml" a todos los equipos involucrados.



La primera información que se puede leer hace referencia a la definición del panel y a la definición de los módulos utilizados en el proceso de fabricación.

Cada panel puede incorporar el número de caras que se desee, descritas todas ellas con el identificador FACE. Normalmente habrá una cara master y una cara esclava (a su espalda), pero también puede darse el caso de que el panel tenga una única cara.

Cada cara puede incorporar el número de tarjetas que se desee, descritas todas ellas con el identificador CARD. Normalmente habrá una tarjeta de alarmas y una tarjeta de errores.

Todos los parámetros monitorizados están en una estructura de árbol,
a disposición de un ADMINISTRADOR DEL SISTEMA
a través de un AGENTE SNMP que va integrado en la aplicación.

Nuestro agente SNMP, además de la labor estándar de contestar las peticiones de información del Administrador, se va a encargar de…

- mantener actualizados ciertos campos del árbol en función del estado de varios otros. Por ejemplo, vincular el Estado General del Display (Ok o Error) en función de la suma de estados de una serie de parámetros concretos.

- lanzar al Administrador cualquier NOTIFICACION-HARDWARE generada por los sensores de las tarjetas. Por ejemplo, temperatura excesiva, puerta abierta, etc.

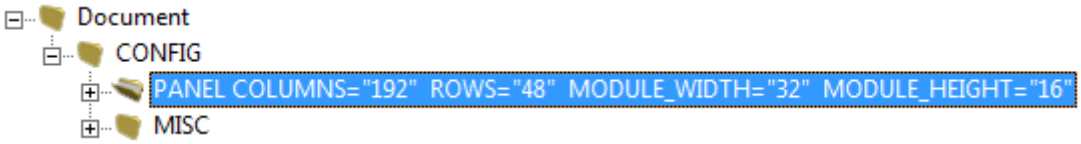
- lanzar al Administrador cualquier NOTIFICACIÓN-SOFTWARE generada por la aplicación en función der una serie de parámetros especificados por el usuario. Por ejemplo, que avise si falla algún led.

- tomas las acciones oportunas tras la llegada de un comando SET cursado desde el Administrador. Por ejemplo, SET = 1 en el identificador del Estado del Termostato para encenderlo remotamente.

- mantener actualizada la información de la ventana o Formulario que permite al usuario conocer los datos más relevantes y cursar las órdenes (mediante BOTONES) que permiten diversas funciones de forma local.

- fijar el brillo del panel en función de las lecturas de la fotocélula de cada cara y en función de la curva de brillo especificada para ese panel (sun, shade, indoor, etc).

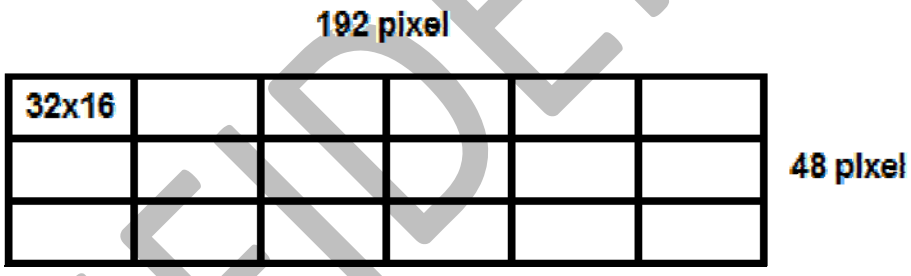
2.1 PANEL



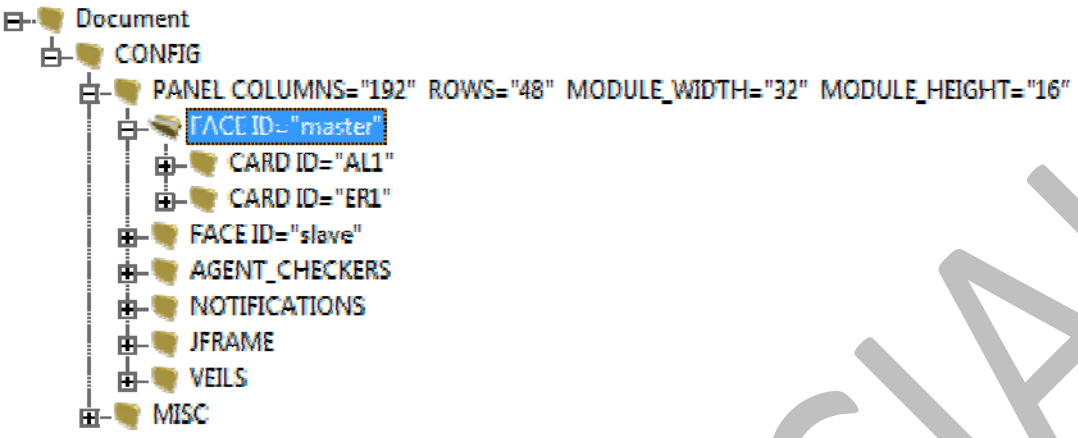
En el ejemplo que ilustramos, el panel va a mostrar el contenido correspondiente a una ventana de 192 columnas por 48 filas de leds.

Como información adicional, se detalla la definición del módulo utilizado para formar cada una de las caras. En este caso, módulos de 32 x 16 píxeles.

Se puede interpretar, entonces, que se necesitan dieciocho módulos en cada cara (tres filas de seis módulos cada una).



I. FACE



Un panel ha de tener necesariamente una cara principal (o **master)**. Normalmente llevará asociada, a su espalda, una cara secundaria (o **esclava**). De esta manera se obtendrá un panel de doble cara que, instalado de forma perpendicular a la dirección del tren, proporcionará información a los viajeros esperando en un andén, a ambos lados del panel.

Al iniciar, para averiguar cuántas caras maneja el panel, la aplicación buscará en el archivo XML de configuración todos los elementos identificados con la palabra clave **FACE**.

Con esta arquitectura de configuración, se podría crear un panel con el número de caras que se desee. Por ejemplo, un panel a cuatro caras en el vestíbulo de una Estación.

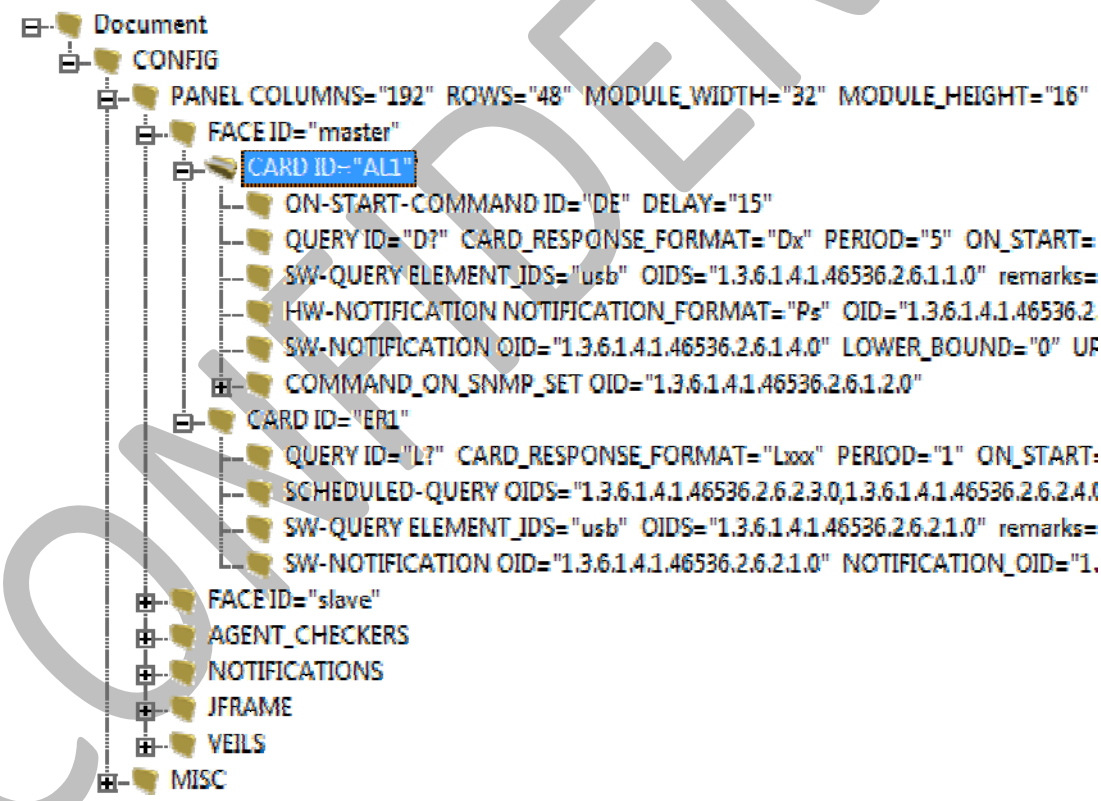
A. CARD

Todas las tarjetas de un panel, tanto las de la cara master como las de la cara slave, se conectan al ordenador que reside en la cara master a través de puertos estándar USB.

La tarjeta de ALARMAS de una cara concreta se encarga de monitorizar los sensores de puerta, golpes, temperatura interior, termostato y ventiladores. También se encarga de encender y apagar la pantalla de esa cara (incluso aunque el panel esté técnicamente encendido) en aquellos casos en los que no interese mostrar el progreso del monitor del PC durante las tareas de arranque o mantenimiento.

La tarjeta de ERRORES de esa misma cara se encarga de monitorizar el sensor de luminosidad y también de detectar los eventuales fallos en leds o módulos.

Reproducimos a continuación un ejemplo simplificado con dos tarjetas CARD que vamos a encontrar en la cara master de nuestro panel: una tarjeta de alarmas (vamos a llamarla AL1) y otra tarjeta de control de errores (ER1)…



INTERPRETACIÓN DE LOS ELEMENTOS CARD

- ***ON-START-COMMAND***

ON-START-COMMAND **ID=”DE”**
DELAY=”15”

Comando (DE) que hay que ejecutar al inicio de la aplicación.

Se puede establecer un **DELAY** en segundos, para que el comando se ejecute con retraso.

En este ejemplo, “DE” encenderá el display transcurridos quince segundos.

- ***QUERY***

QUERY **ID="D?"**
CARD_RESPONSE_FORMAT="Dx"
PERIOD="5"
ON_START="1"
OIDS="1.3.6.1.4.1.46536.2.6.1.2.0"
remarks="DISPLAY ON/OFF"

Comando que ejecuta una consulta (**D?**) al hardware, a intervalos regulares (**PERIOD**).

Se establece el formato de la respuesta como una ‘**D**’ seguida de un número (**x**).

Además de las consultas periódicas, se puede realizar una consulta inicial, al arrancar la aplicación. **ON_START** indica cuántos segundos hay que esperar tras el arranque para ejecutar una consulta inicial. Un número de segundos =”-1” deshabilita la ejecución inicial.

Se especifica el identificador **OID** del Agente (la “hoja” con información en el árbol de datos) que se verá actualizado con la respuesta.

En este ejemplo, QUERY preguntará cada 5 segundos por el estado encendido/apagado del panel. Además, hará la misma pregunta al iniciar la aplicación, transcurrido un segundo.

Otros formatos de respuesta: ‘**s**’ = símbolo
‘**x**’ = carácter numérico
‘**a**’ = carácter alfabético
‘**y**’ = cadena de números

Lista de comandos QUERY…

D?	pregunta por el estado encendido/apagado del panel.
P?	pregunta por el estado abierta/cerrada de la puerta.
T?	pregunta por la temperatura interior del panel.
S?	pregunta por el estado encendido/apagado del termostato.
V?	pregunta por el estado girando/parado de los ventiladores.
L?	pregunta por el nivel de luz detectado por la fotocélula.
E?	pregunta cuántos leds o módulos fallan.

- ***SCHEDULED-QUERY***

SCHEDULED-QUERY **ID="E?"**
AT_SCHEDULED_TIME="03:01"
ELEMENT_IDS=”faultyLeds, faultyModules"
CARD_RESPONSE_FORMAT="Ey9216"
OIDS= ”1.3.6.1.4.1.46536.2.6.2.3.0,
1.3.6.1.4.1.46536.2.6.2.4.0"
ON_START="2"

Comando que ejecuta una consulta (**ID**) al hardware cada día, a una hora concreta (**AT_SCHEDULED_TIME**). Al igual que en una QUERY, hay que establecer el formato de respuesta (**CARD_RESPONSE_FORMAT**), el OID (**OIDS**) del Agente que se va a actualizar y si hay que ejecutar el comando al inicio (**ON_START**) de la aplicación.

Además, se permite que la misma consulta actualice a la vez más de un elemento en el Agente, cada uno con información diferente. Para ello es necesario dar una identidad a cada uno de estos elementos (**ELEMENT_IDS**) y vincularlo con su OID correspondiente. Hay que indicar tantos OIDs como ELEMENT_IDs. De tal forma que el primer **OIDS** se identifique con primer ELEMENT_IDS, el segundo OIDS con el segundo ELEMENT_IDS y así sucesivamente.

En este ejemplo, SCHEDULED-QUERY preguntará cada día por el estado de los leds a las 03:01 horas de la mañana. Antes, también lo preguntará dos segundos después de haber iniciado el programa. De ese dato extraerá el número de leds con error y el número de placas con error y pondrá ambos datos a disposición del Agente, en sus correspondientes OIDs.

- ***SOFTWARE-QUERY***

SW-QUERY **ELEMENT_IDS=”usb"**
OIDS= ” 1.3.6.1.4.1.46536.2.6.1.1.0"

El usuario quiere monitorizar un elemento, que identifica con el nombre “usb”, y genera un vínculo entre este elemento de la aplicación SOFTWARE y un OID del Agente, de manera que, por ejemplo, se pueda lanzar una notificación al Administrador cuando el puerto deje de estar disponible.

- ***HARDWARE-NOTIFICATION***

HW-NOTIFICATION **NOTIFICATION_FORMAT=”Ps"**
OIDS= ” 1.3.6.1.4.1.46536.2.6.1.1.0"
MESSAGE=”Door Open”

El hardware genera ciertas notificaciones de manera automática (por ejemplo, puerta abierta). Por ello, se necesita indicar el formato del mensaje que se va a recibir del hardware y vincularlo con el OID que se corresponde con dicha notificación.

- ***SOFTWARE-NOTIFICATION***

SW-NOTIFICATION **OID="1.3.6.1.4.1.46536.2.6.1.1.0"**
NOTIFICATION_ OID="1.3.6.1.4.1.46536.2.6.3.2.0"
ELEMENT=”masterUsbAlarmsNotification"
CONDITION="STRING_VALUE_ERROR_CONDITION"
STRING_VALUE_ERROR_CONDITION=""

Cualquier elemento del Agente (OID) puede generar una NOTIFICACIÓN (representada por su identificador NOTIFICATION_ OID) si su valor cumple una condición.

En el ejemplo, se da de alta una notificación software (SW-NOTIFICATION), que monitoriza el USB de la tarjeta de alarmas de la cara master (lo sabemos por el OID), y lanzará una notificación al Administrador (a su NOTIFICATION_OID) si el valor del elemento cumple la condición de coincidir con el valor de la STRING _VALUE_ERROR_CONDITION. Es decir, e este caso, si el valor del elemento está vacío.

Puede haber otras condiciones…

Lista de posibles condiciones que generarán notificación:

- **STRING_VALUE_ERROR_CONDITION**: si el valor del elemento del Agente coincide con el valor declarado, se genera la notificación.

- **BOUNDS**: se define un rango de valores válidos declarando **UPPER_BOUND** y **LOWER_BOUND**. Cualquier valor del elemento del Agente fuera de estos límites, generará notificación.

- **VALUE_MUST_MATCH_TO**: el valor del elemento OID debe coincidir con el de todos los elementos declarados en la condición. Si hay alguna discrepancia, se genera la notificación. Hemos incorporado, además, el atributo **RETRIES**, que permite reintentar varias veces seguidas el cumplimiento de la condición antes de generar la notificación.

  Por ejemplo, el valor del termostato debe ser siempre igual al de los ventiladores, dado que si el termostato está encendido, los ventiladores deben girar. En este caso el atributo RETRIES es imprescindible por la inercia que tienen los ventiladores, tanto al comenzar a girar como al parar.

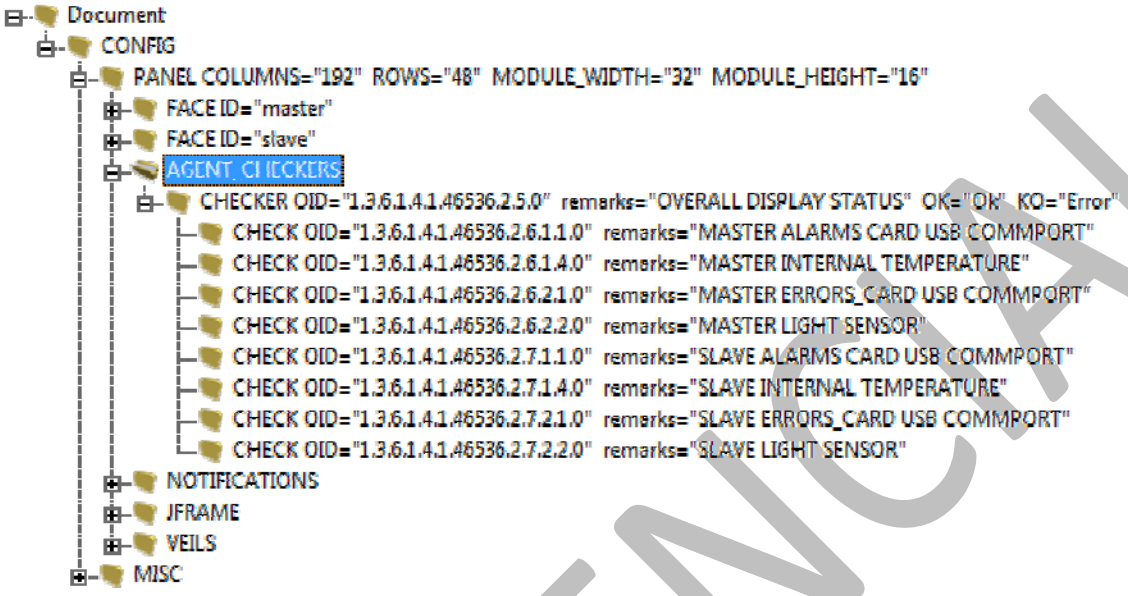
- ***COMMAND-ON-SNMP-SET***

```
COMMAND-ON-SNMP-SET    OID="1.3.6.1.4.1.46536.2.6.1.2.0"
                       COMMAND
                              LEAF_NODE_VALUE=”0”
                              ID=”DA”
                              remarks=”APAGAR DISPLAY”
                       COMMAND
                              LEAF_NODE_VALUE=”1”
                              ID=”DE”
                              remarks=”ENCENDER DISPLAY”
```

El usuario puede establecer las acciones que se desencadenarán al recibir desde el Administrador un SET sobre un OID concreto del Agente, con alguno de los valores contemplados en cualquiera de los elementos COMMAND descritos.

En el ejemplo, se apagará el display (DA) si se recibe en ese OID un SET con valor “1”.

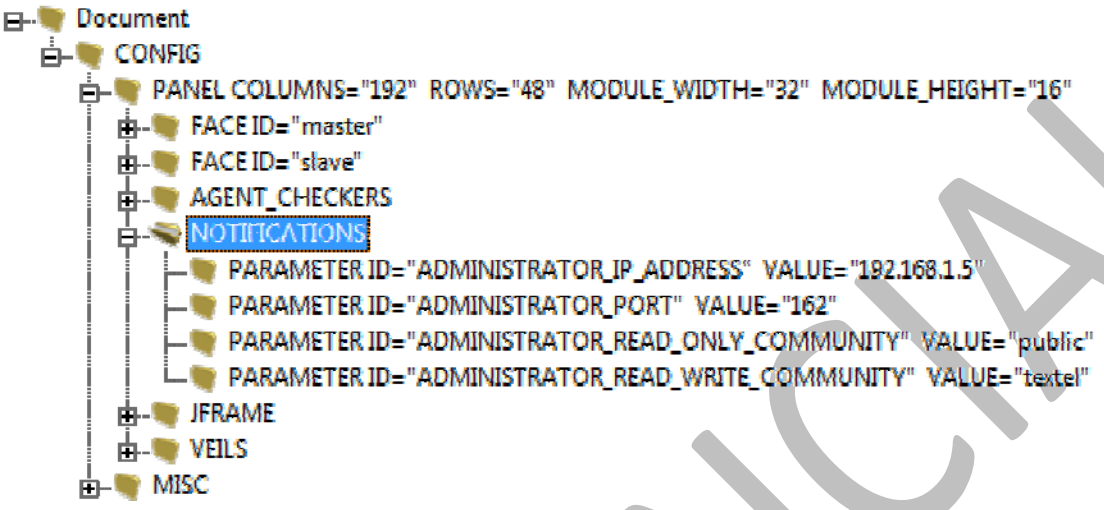
II. AGENT_CHECKERS



El usuario puede declarar el número de AGENT_CHECKERS que desee. Para cada CHECKER hay que declarar el OID que recogerá el valor OK o KO resultado de analizar todos los OID de los CHECK declarados a continuación para ese CHECKER.

En el ejemplo, el Administrador puede consultar el OID: 1.3.6.1.4.1.46536.2.5.0 para conocer el estado general del PANEL, que estará OK si también lo están todos los elementos de las dos caras (MASTER y SLAVE) que se declaran: usb de la tarjeta de alarmas, temperatura interna, usb de la tarjeta de errores y sensor de luz.

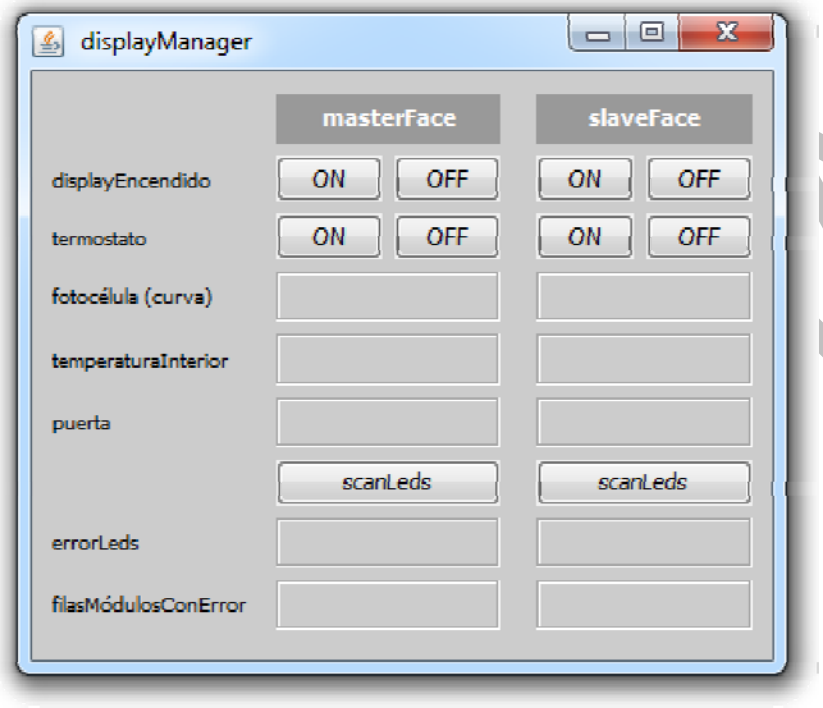
III. NOTIFICATIONS



En este elemento se declaran los parámetros necesarios (dirección IP, puerto y communities) para establecer la comunicación con el usuario. Tanto para enviar NOTIFICACIONES como para permitir los comandos remotos COMMAND-ON-SNMP-SET. Estos últimos deben llegar con la comunidad que tiene acceso a escritura.

IV. JFRAME

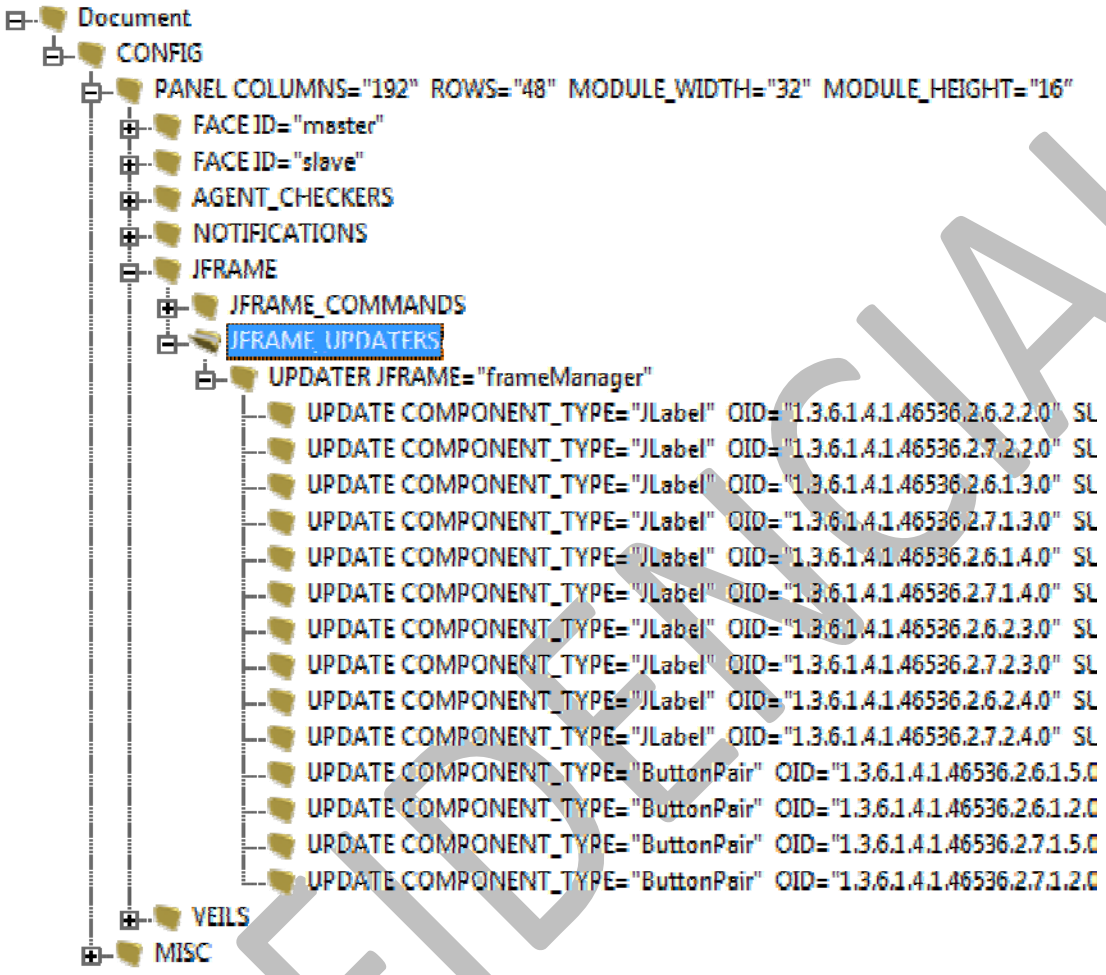
La aplicación pone a disposición del usuario un formulario que permite enviar comandos en modo local a cualquiera de las tarjetas CARD de cualquier cara FACE.

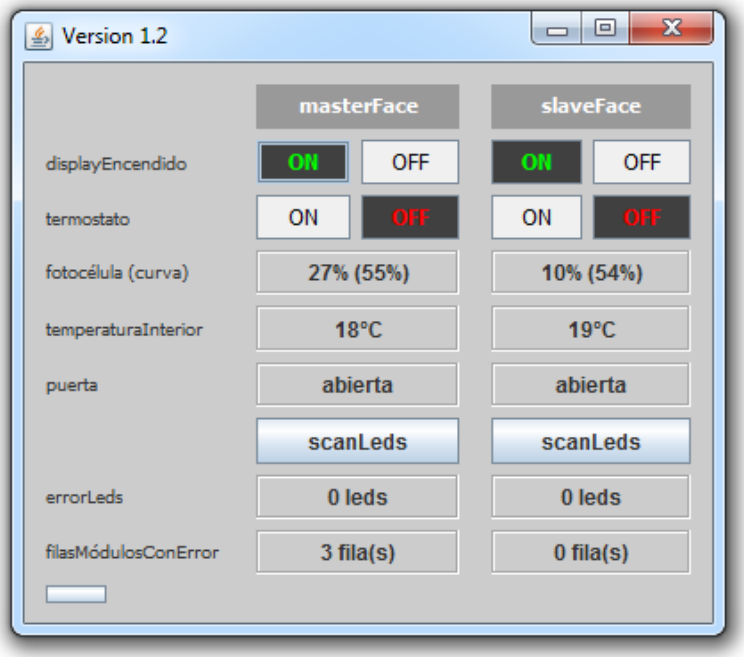


El elemento JFRAME_COMMANDS del fichero XML a continuación, declara los COMMAND_BUTTON que adjudican comandos concretos a los diez botones del formulario.

```
Document
  CONFIG
    PANEL COLUMNS="192" ROWS="48" MODULE_WIDTH="32" MODULE_HEIGHT="16"
      FACE ID="master"
      FACE ID="slave"
      AGENT_CHECKERS
      NOTIFICATIONS
      JFRAME
        JFRAME_COMMANDS
          COMMAND_BUTTON ID="cmdMasterScanLeds" CARD_ID="ER1" FACE_ID="master" COMMAND="EI"
          COMMAND_BUTTON ID="cmdSlaveScanLeds" CARD_ID="ER2" FACE_ID="slave" COMMAND="EI"
          COMMAND_BUTTON ID="cmdMasterTermostatoOn" CARD_ID="AL1" FACE_ID="master" COMMAND="SE"
          COMMAND_BUTTON ID="cmdMasterTermostatoOff" CARD_ID="AL1" FACE_ID="master" COMMAND="SA"
          COMMAND_BUTTON ID="cmdSlaveTermostatoOn" CARD_ID="AL2" FACE_ID="slave" COMMAND="SE"
          COMMAND_BUTTON ID="cmdSlaveTermostatoOff" CARD_ID="AL2" FACE_ID="slave" COMMAND="SA"
          COMMAND_BUTTON ID="cmdMasterDisplayOn" CARD_ID="AL1" FACE_ID="master" COMMAND="DE"
          COMMAND_BUTTON ID="cmdMasterDisplayOff" CARD_ID="AL1" FACE_ID="master" COMMAND="DA"
          COMMAND_BUTTON ID="cmdSlaveDisplayOn" CARD_ID="AL2" FACE_ID="slave" COMMAND="DE"
          COMMAND_BUTTON ID="cmdSlaveDisplayOff" CARD_ID="AL2" FACE_ID="slave" COMMAND="DA"
        JFRAME_UPDATERS
      VEILS
    MISC
```

Por otro lado, las ventanas JLABEL del formulario que aparecen recuadradas en gris claro se van a rellenar con los datos declarados en el elemento JFRAME_UPDATERS del fichero XML, según se vayan actualizando en el árbol de datos del Agente SNMP.





V. VEILS

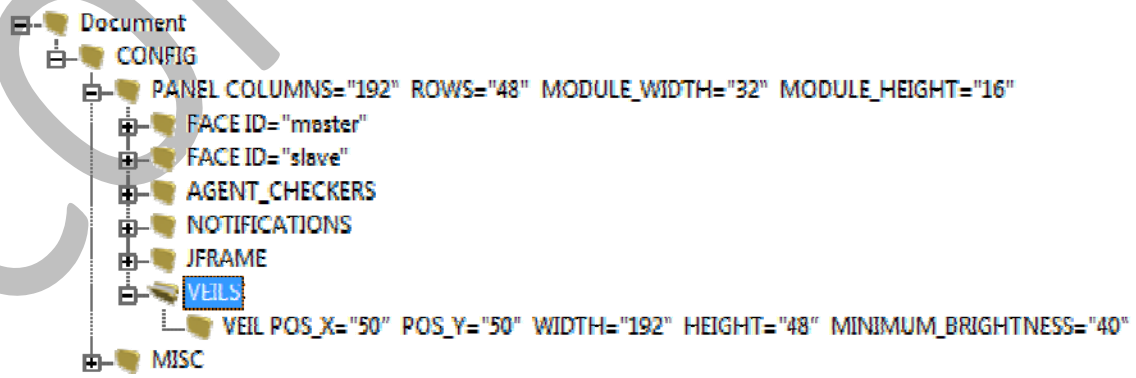
El brillo de la información que muestra el PANEL se regula mediante un "velo" más o menos opaco, en función de las consecutivas lecturas de los niveles de luz ambiental detectados por las fotocélulas.

Nuestro panel de leds actúa como un segundo monitor (clonado) del PC que está en la cara master. Se reproduce, píxel a píxel, el contenido que el Administrador muestra en una región preestablecida del monitor, que tiene un tamaño equivalente, en píxeles, a la definición de cada cara del panel. Es ahí donde la aplicación también ubica un velo que "flota" sobre los contenidos para controlar el brillo percibido.

Básicamente, en lugar de actuar sobre la electrónica, simplificamos las operaciones y actuamos sobre el velo, variando su índice de opacidad de acuerdo a la lectura menos favorable de las que podamos monitorizar, y de acuerdo también a la curva de brillo designada para la ubicación concreta del panel dentro de la Estación (sol, sombra, interior, etc).

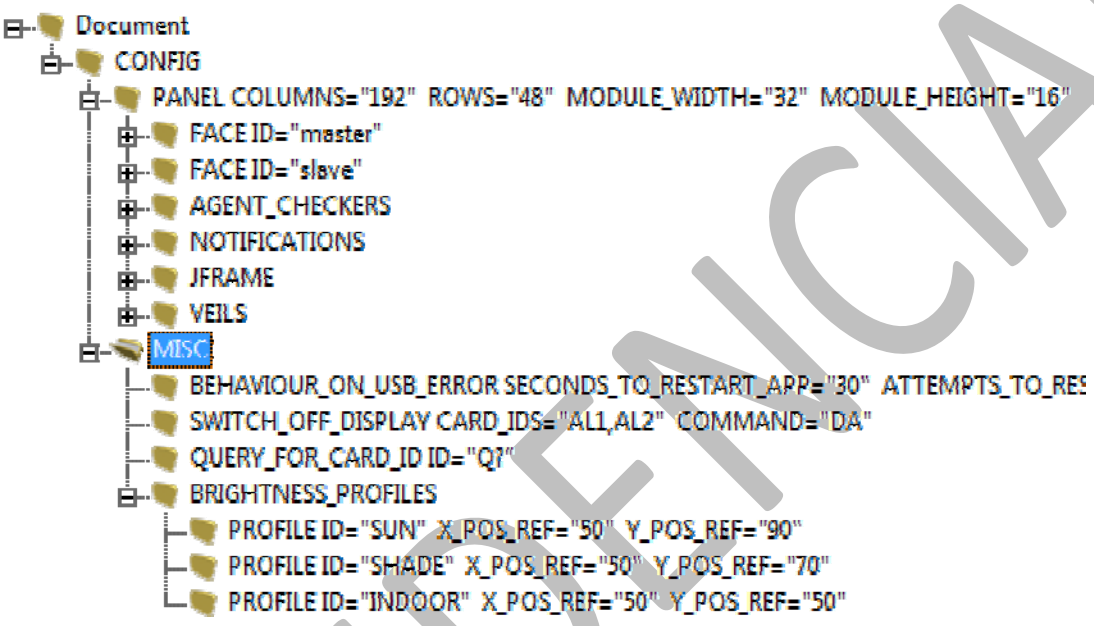
Se puede crear velos (normalmente uno) detallando…

- su ubicación en el escritorio mediante las coordenadas (X,Y) de su esquina superior izquierda.
- su tamaño ancho y alto.
- el brillo mínimo que va a mostrar, incluso aunque las lecturas de las fotocélulas sean menores.
- el factor de conversión que hay que aplicar para obtener las lecturas en un rango de 0-100 (porque la fotocélula declara el brillo detectado en un rango de 1-256).
- los OIDs del Agente de donde se va a tomar la información.



2.2 MISCELLANEA

Por último, hay una serie de parámetros que no son exclusivos de los elementos descritos hasta el momento, y que se detallan a continuación.



- **Comportamiento en caso de detectar error en USB**

  Número de segundos antes de reiniciar la aplicación. Durante este tiempo se intenta reconfigurar los puertos cada vez que se ejecuta un Query (en cualquiera de sus modalidades: simple, scheduled o software).

  Número de intentos de reiniciar la aplicación antes de reiniciar el PC. Si persiste el error después de haber reiniciado la aplicación varias veces, entonces se reiniciará el ordenador completamente.

  Número de intentos de reiniciar el PC antes de “dormir” la aplicación. Si persiste el error después de haber reiniciado el PC varias veces, entonces se notificará y se entrará en un estado de pausa en el que se dejará de enviar notificaciones y solamente se estará monitorizando la aparición de USBs de nuevo. En ese caso, la aplicación se despierta para volver a ser completamente operativa.

- **Tarjetas encargadas de apagar el display**

  Para cuando así le sea requerido, la aplicación software debe saber qué tarjetas son capaces de apagar el display, y mediante qué comando.

  En este elemento se listan, separadas por comas, las tarjetas que son capaces de realizar ese cometido, y se especifica el comando 'DA' que deben ejecutar para apagar el display.

- **Comando para preguntar a una tarjeta su identificación**

  Al iniciar la aplicación, una de las primeras tareas que hay que realizar es averiguar de cuántos puertos USB dispone el sistema, y preguntar después, por cada uno de esos puertos, si hay alguna tarjeta conectada. El comando para preguntar a una tarjeta su identificador es 'Q?' .

  Con esas acciones podremos montar un mapa de identificadores de tarjetas, con los puertos a los que están conectadas.

- **Perfiles de brillo**

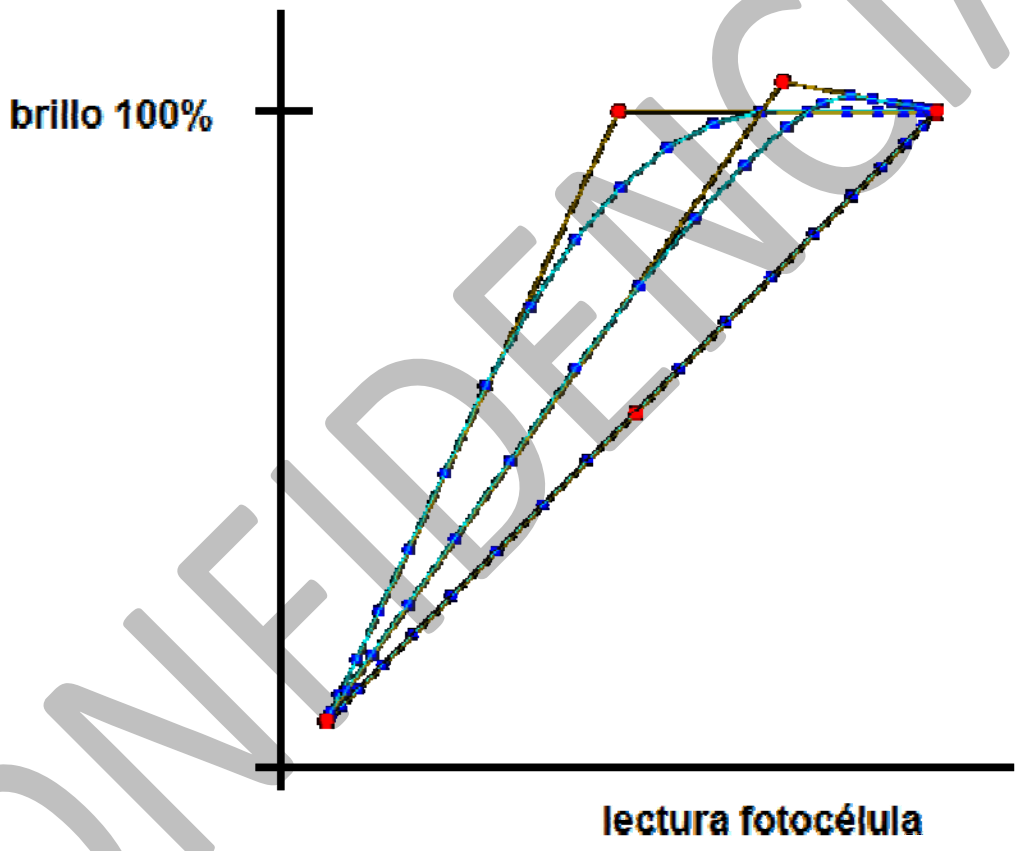
  El usuario puede crear los perfiles de brillo que desee. No obstante, inicialmente se entregan tres para ubicaciones estándar: sol, sombra e interior.

  Para un interior, el brillo del panel puede ser proporcional a la lectura de la fotocélula. En cambio, en un exterior, con abundante luz natural en el ambiente, necesitaremos que valores intermedios de lectura se traduzcan en brillos altos.

  Básicamente, lo que vamos a hacer con un perfil es obligar a la curva de brillo a pasar por un punto intermedio en el plano que modificará sustancialmente la asignación de brillo en función de la lectura de la fotocélula.

  En los perfiles existentes, el usuario puede modificar las coordenadas X e Y del punto intermedio. Si prefiere crear uno nuevo, deberá adjudicar un nombre y, seguidamente, las coordenadas del punto.

  En la página siguiente se puede ver un ejemplo de distintos comportamientos de brillo para una misma lectura.



3. FICHERO MIB

La estructura “árbol de datos” que la aplicación tiene que mantener actualizada en el equipo está definida en el MIB FILE con nombre ‘TEXTEL-TLIA.mib’.

Textel Marimón S.A. está registrada en el IANA (Internet Assigned Numbers Authority) con el número de empresa privada 46.536. Esta entidad es la que supervisa la asignación global de direcciones IP, sistemas autónomos, servidores raíz de nombres de dominio DNS y otros recursos relativos a los protocolos de Internet.

Todos los componentes monitorizados en los paneles se derivan del prefijo…

iso.org.dod.internet.private.enterprise.textel-marimon. (1.3.6.1.4.1.46536.)

Así, por ejemplo, el valor de luz ambiental detectado por el sensor de la cara máster está localizado en el Identificador de Objeto (OID)…

MasterLightSensor OID : 1.3.6.1.4.1.46536.2.6.2.2

Este fichero MIB también debe residir en el Administrador, ya que le es necesario para poder consultar o modificar cualquier variable de un equipo de la red, así como para predefinir las notificaciones que puede recibir desde cualquiera de los agentes conectados.

A continuación se detalla su contenido…

TEXTEL-TLIA.mib

```
-- ************************************************************************
-- *
-- * Copyright (c) 2015 Textel Marimón s.a. Inc. All Rights Reserved.
-- *
-- * Module Name:  TEXTEL-TLIA
-- *
-- * Description:  DEFINE OBJETOS PARA MANEJAR TELEINDICADORES
-- *
-- ************************************************************************

TEXTEL-TLIA DEFINITIONS ::= BEGIN


        IMPORTS
        NOTIFICATION-TYPE, OBJECT-TYPE, MODULE-IDENTITY,
        enterprises, Integer32
                FROM SNMPv2-SMI
        NOTIFICATION-GROUP, OBJECT-GROUP, MODULE-COMPLIANCE
                FROM SNMPv2-CONF
        RowStatus, DisplayString
                FROM SNMPv2-TC;


textel-Marimon   OBJECT IDENTIFIER ::= { enterprises 46536 }


        nombreCompleto OBJECT-TYPE
        SYNTAX OCTET STRING
        ACCESS read-only
        STATUS current
        DESCRIPTION
                "Nombre del fabricante del display."
        ::= { textel-Marimon 1 }


display OBJECT IDENTIFIER ::= { textel-Marimon 2 }


        lane OBJECT-TYPE
                SYNTAX OCTET STRING
                ACCESS read-only
                STATUS current
                DESCRIPTION
                        "Linea donde está instalado el teleindicador."
                ::= { display 1 }

        station OBJECT-TYPE
                SYNTAX OCTET STRING
                ACCESS read-only
                STATUS current
                DESCRIPTION
                        "Estación."
                ::= { display 2 }

        platform OBJECT-TYPE
                SYNTAX INTEGER
                ACCESS read-only
                STATUS current
                DESCRIPTION
                        "Andén."
                ::= { display 3 }

        position OBJECT-TYPE
                SYNTAX INTEGER
                ACCESS read-only
                STATUS current
                DESCRIPTION
                        "Número de orden en el andén."
                ::= { display 4 }

        displayGeneralStatus OBJECT-TYPE
                SYNTAX OCTET STRING
                ACCESS read-only
                STATUS current
                DESCRIPTION
                        "Estado general del teleindicador (Ok / Error)."
                ::= { display 5 }
```

```
master OBJECT IDENTIFIER ::= { display 6 }

        masterAlarmas OBJECT IDENTIFIER ::= { master 1 }

                masterUsbTarjetaAlarmas OBJECT-TYPE
                        SYNTAX OCTET STRING
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: estado USB tarjeta alarmas (Ok / Error)."
                        ::= { masterAlarmas 1 }

                masterDisplayEncendido OBJECT-TYPE
                        SYNTAX Integer32
                        ACCESS read-write
                        STATUS current
                        DESCRIPTION
                                "Master: encendido (0 = OFF / 1 = ON)."
                        ::= { masterAlarmas 2 }

                masterPuerta OBJECT-TYPE
                        SYNTAX OCTET STRING
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: estado puerta (Abierta / Cerrada)."
                        ::= { masterAlarmas 3 }

                masterTemperaturaInterior OBJECT-TYPE
                        SYNTAX Integer32
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: temperatura interior (°C)"
                        ::= { masterAlarmas 4 }

                masterTermostato OBJECT-TYPE
                        SYNTAX Integer32
                        ACCESS read-write
                        STATUS current
                        DESCRIPTION
                                "Master: estado termostato (0 = OFF / 1 = ON)."
                        ::= { masterAlarmas 5 }

                masterVentilador1 OBJECT-TYPE
                        SYNTAX OCTET STRING
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: estado del ventilador 1 (Ok / Error)."
                        ::= { masterAlarmas 6 }

                masterVentilador2 OBJECT-TYPE
                        SYNTAX OCTET STRING
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: estado del ventilador 2 (Ok / Error)."
                        ::= { masterAlarmas 7 }

        masterVentilador3 OBJECT-TYPE
                        SYNTAX OCTET STRING
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: estado del ventilador 3 (Ok / Error)."
                        ::= { masterAlarmas 8 }

                masterVentilador4 OBJECT-TYPE
                        SYNTAX OCTET STRING
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: estado del ventilador 4 (Ok / Error)."
                        ::= { masterAlarmas 9 }


        masterErrores OBJECT IDENTIFIER ::= { master 2 }

                masterUsbTarjetaErrores OBJECT-TYPE
                        SYNTAX OCTET STRING
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: estado USB tarjeta errores (Ok / Error)."
                        ::= { masterErrores 1 }
```

```
                masterNivelFotocelula OBJECT-TYPE
                        SYNTAX Integer32
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: nivel fotócelula (1 - 100 / 0 = error)."
                        ::= { masterErrores 2 }

                masterLeds OBJECT-TYPE
                        SYNTAX Integer32
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: número de leds con error"
                        ::= { masterErrores 3 }

                masterModulos OBJECT-TYPE
                        SYNTAX Integer32
                        ACCESS read-only
                        STATUS current
                        DESCRIPTION
                                "Master: número de filas de módulos con error"
                        ::= { masterErrores 4 }

        masterNotificaciones OBJECT IDENTIFIER ::= { master 3 }


                masterDisplayApagado NOTIFICATION-TYPE
                        STATUS current
                        DESCRIPTION
                                "Master: display apagado"
                        ::= { masterNotificaciones 1 }

                masterErrorUsbAlarms NOTIFICATION-TYPE
                        STATUS current
                        DESCRIPTION
                                "Master: error Usb"
                        ::= { masterNotificaciones 2 }

                masterErrorUsbErrors NOTIFICATION-TYPE
                        STATUS current
                        DESCRIPTION
                                "Master: error Usb"
                        ::= { masterNotificaciones 3 }

                masterPuertaAbierta NOTIFICATION-TYPE
                        STATUS current
                        DESCRIPTION
                                "Master: puerta abierta"
                        ::= { masterNotificaciones 4 }

                masterTemperaturaExcedida NOTIFICATION-TYPE
                        STATUS current
                        DESCRIPTION
                                "Master: temperatura excedida"
                        ::= { masterNotificaciones 5 }

                masterErrorVentiladores NOTIFICATION-TYPE
                        STATUS current
                        DESCRIPTION
                                "Master: fallo en ventiladores"
                        ::= { masterNotificaciones 6 }

                masterErrorLeds NOTIFICATION-TYPE
                        STATUS current
                        DESCRIPTION
                                "Master: leds con error"
                        ::= { masterNotificaciones 7 }

                masterErrorModulos NOTIFICATION-TYPE
                        STATUS current
                        DESCRIPTION
                                "Master: módulos Led con error"
                        ::= { masterNotificaciones 8 }

                masterImpacto NOTIFICATION-TYPE
                        STATUS current
                        DESCRIPTION
                                "Master: impacto (en Gs)"
                        ::= { masterNotificaciones 9 }

                masterErrorFotocelula NOTIFICATION-TYPE
                        STATUS current
                        DESCRIPTION
                                "Master: fotocélula con error"
                        ::= { masterNotificaciones 10 }

slave OBJECT IDENTIFIER ::= { display 7 }

        slaveAlarmas OBJECT IDENTIFIER ::= { slave 1 }
```

```
slaveUsbTarjetaAlarmas OBJECT-TYPE
        SYNTAX OCTET STRING
        ACCESS read-only
        STATUS current
        DESCRIPTION
                "Slave: estado USB tarjeta alarmas (Ok / Error)."
        ::= { slaveAlarmas 1 }

slaveDisplayEncendido OBJECT-TYPE
        SYNTAX Integer32
        ACCESS read-write
        STATUS current
        DESCRIPTION
                "Slave: encendido (0 = OFF / 1 = ON)."
        ::= { slaveAlarmas 2 }

slavePuerta OBJECT-TYPE
        SYNTAX OCTET STRING
        ACCESS read-only
        STATUS current
        DESCRIPTION
                "Slave: estado puerta (Abierta / Cerrada)."
        ::= { slaveAlarmas 3 }

slaveTemperaturaInterior OBJECT-TYPE
        SYNTAX Integer32
        ACCESS read-only
        STATUS current
        DESCRIPTION
                "Slave: temperatura interior (°C)"
        ::= { slaveAlarmas 4 }

slaveTermostato OBJECT-TYPE
        SYNTAX Integer32
        ACCESS read-write
        STATUS current
        DESCRIPTION
                "Slave: estado termostato (0 = OFF / 1 = ON)."
        ::= { slaveAlarmas 5 }

slaveVentilador1 OBJECT-TYPE
        SYNTAX OCTET STRING
        ACCESS read-only
        STATUS current
        DESCRIPTION
                "Slave: estado del ventilador 1 (Ok / Error)."
        ::= { slaveAlarmas 6 }

slaveVentilador2 OBJECT-TYPE
        SYNTAX OCTET STRING
        ACCESS read-only
        STATUS current
        DESCRIPTION
                "Slave: estado del ventilador 2 (Ok / Error)."
        ::= { slaveAlarmas 7 }

slaveVentilador3 OBJECT-TYPE
        SYNTAX OCTET STRING
        ACCESS read-only
        STATUS current
        DESCRIPTION
                "Slave: estado del ventilador 3 (Ok / Error)."
        ::= { slaveAlarmas 8 }

slaveVentilador4 OBJECT-TYPE
        SYNTAX OCTET STRING
        ACCESS read-only
        STATUS current
        DESCRIPTION
                "Slave: estado del ventilador 4 (Ok / Error)."
        ::= { slaveAlarmas 9 }


slaveErrores OBJECT IDENTIFIER ::= { slave 2 }


slaveUsbTarjetaErrores OBJECT-TYPE
        SYNTAX OCTET STRING
        ACCESS read-only
        STATUS current
        DESCRIPTION
                "Slave: estado USB tarjeta errores (Ok / Error)."
        ::= { slaveErrores 1 }

slaveNivelFotocelula OBJECT-TYPE
        SYNTAX Integer32
        ACCESS read-only
        STATUS current
        DESCRIPTION
                "Slave: nivel fotócelula (1 - 100 / 0 = error)."
        ::= { slaveErrores 2 }
```

```
        slaveLeds OBJECT-TYPE
                SYNTAX Integer32
                ACCESS read-only
                STATUS current
                DESCRIPTION
                        "Slave: número de leds con error"
                ::= { slaveErrores 3 }

        slaveModulos OBJECT-TYPE
                SYNTAX Integer32
                ACCESS read-only
                STATUS current
                DESCRIPTION
                        "Slave: número de filas de módulos con error"
                ::= { slaveErrores 4 }


slaveNotificaciones OBJECT IDENTIFIER ::= { slave 3 }


        slaveDisplayApagado NOTIFICATION-TYPE
                STATUS current
                DESCRIPTION
                        "Slave: display apagado"
                ::= { slaveNotificaciones 1 }

        slaveErrorUsbAlarms NOTIFICATION-TYPE
                STATUS current
                DESCRIPTION
                        "Slave: error Usb"
                ::= { slaveNotificaciones 2 }

        slaveErrorUsbErrors NOTIFICATION-TYPE
                STATUS current
                DESCRIPTION
                        "Slave: error Usb"
                ::= { slaveNotificaciones 3 }

        slavePuertaAbierta NOTIFICATION-TYPE
                STATUS current
                DESCRIPTION
                        "Slave: puerta abierta"
                ::= { slaveNotificaciones 4 }

        slaveTemperaturaExcedida NOTIFICATION-TYPE
                STATUS current
                DESCRIPTION
                        "Slave: temperatura excedida"
                ::= { slaveNotificaciones 5 }

        slaveErrorVentiladores NOTIFICATION-TYPE
                STATUS current
                DESCRIPTION
                        "Slave: fallo en ventiladores"
                ::= { slaveNotificaciones 6 }

        slaveErrorLeds NOTIFICATION-TYPE
                STATUS current
                DESCRIPTION
                        "Slave: leds con error"
                ::= { slaveNotificaciones 7 }

        slaveErrorModulos NOTIFICATION-TYPE
                STATUS current
                DESCRIPTION
                        "Slave: módulos Led con error"
                ::= { slaveNotificaciones 8 }

        slaveImpacto NOTIFICATION-TYPE
                STATUS current
                DESCRIPTION
                        "Slave: impacto (en Gs)"
                ::= { slaveNotificaciones 9 }

        slaveErrorFotocelula NOTIFICATION-TYPE
                STATUS current
                DESCRIPTION
                        "Slave: fotocélula con error"
                ::= { slaveNotificaciones 10 }


END
```