



---

# INSTITUTO POLITECNICO NACIONAL

## ESCUELA SUPERIOR DE INGENIERIA MECANICA Y ELECTRICA UNIDAD ZACATENCO

### **Carrera:**

Ingeniería en Comunicaciones y Electrónica con Especialidad en  
Computación

### **Tema de Tesis:**

Implementación de Sistema de Control de Acceso Mediante Huella Digital

### **Alumno:**

Gabriel Ibarra Romero

### **Asesores:**

Ing. Sebastián Villa Cruz

M en C. José Luís Bravo León

## **Agradecimientos**

A mi madre por haberme dado la oportunidad de estudiar una carrera.

A mi esposa y a mis hermanos por motivarme e incitarme a dar el último paso y terminar mi tesis para así obtener un título de ingeniero.

A todas esas personas que se atreven a hacer realidad sus sueños y a lograr cosas que otros ni siquiera nunca se hubiesen imaginado, y como resultado hacen que el mundo cambie.

A todas esas personas que se esfuerzan por lograr sus metas, sin importar lo extrañas o complejas que puedan parecer.

A todas esas personas que son una fuente de inspiración para todos aquellos que se atreven a soñar.

# Índice

Objetivo	i
Justificación	ii
Presentación	iii
Capítulo 1. Introducción	1
Capítulo 2. Descripción del Problema	4
Capítulo 3. Posibles Soluciones	7
Capítulo 4. Justificación de la Solución	11
Capítulo 5. Implementación	14
5.1 Funcionamiento General de la Aplicación	14
5.2 La Base de Datos del Sistema	16
5.3 El Servicio Web	21
5.4 La Base de Datos de las Terminales	27
5.5 El Hardware de las Terminales	28
5.6 El Software de las Terminales	34
Capítulo 6. Resultados Obtenidos	102
Capítulo 7. Consideraciones Técnico-Económicas	110
Capítulo 8. Conclusiones	112
Anexos	
A. Rutinas del Servicio Web	115
B. Rutinas de la Aplicación de las Terminales	118
Referencias Bibliográficas	129

## **Objetivo**

El objetivo del trabajo aquí redactado es diseñar e implementar un sistema de control de acceso funcional mediante la tecnología de reconocimiento de huella digital.

## Justificación

Este proyecto se realizó porque aunque en el mercado existen muchos sistemas de control de acceso, ninguno se puede personalizar lo suficiente para cubrir exactamente con la necesidad de la empresa que lo solicitó, llamada “House of Fuller”.

Algunos ejemplos de sistemas que se pueden encontrar en el mercado son los siguientes:

- **Secugen Hamster**
- **FINGER-007 Fingerprint Identification (Proximity / PIN) Time & Attendance Access Controller**
- ***Bio-OFFICE OA100 Fingerprint T&A System***

En general los precios de estos sistemas son accesibles aún considerando que se necesitará instalar nueve dispositivos para controlar nueve accesos, pero ninguno de estos sistemas cumple exactamente con lo que la empresa solicitó.

Además la empresa ya adquirió hardware con un valor total aproximado de \$31,350\* para implementar el sistema, y al desarrollar este proyecto se aprovechará ese hardware casi en su totalidad.

---

\* Precios en dólares americanos

## **Presentación**

En este trabajo se describe la forma en que se realizó una implementación de un sistema de control de acceso mediante la tecnología de huella digital, rescatando casi totalmente un hardware que estaba próximo a desecharse.

Este trabajo consta de ocho capítulos, iniciando desde un entorno introductorio a los sistemas de seguridad y a la empresa que lo solicitó, hasta los resultados obtenidos después de su implantación.

**Capítulo 1. Introducción:** Se proporciona un entorno histórico de los sistemas de seguridad y un panorama general del trabajo descrito en esta tesis.

**Capítulo 2. Descripción del Problema:** Se explica un problema que tuvo una empresa y que dio origen al desarrollo del sistema que aquí se describe.

**Capítulo 3. Posibles Soluciones:** Se explican de una forma resumida las posibles alternativas que existían para resolver el problema que se le presentó a la empresa.

**Capítulo 4. Justificación de la Solución:** Se explica a grandes rasgos en que consistió la solución elegida y el porqué se eligió.

**Capítulo 5. Implementación:** Se explica con todo detalle los diferentes componentes del sistema que se desarrolló. Estos se enlistan a continuación:

- a. Funcionamiento general de la aplicación
- b. La base de datos del sistema
- c. El servicio Web
- d. La base de datos de las terminales
- e. El hardware de las terminales
- f. El software de las terminales

**Capítulo 6. Conclusiones:** Se comentan algunas dificultades que se tuvieron que solucionar al momento de implementar el sistema.

**Capítulo 7. Resultados Obtenidos:** Se proporciona una visión general de cómo funciona el sistema una vez terminado.

**Capítulo 8. Consideraciones Técnico-Económicas:** Se proporcionan detalladamente las características técnicas de los equipos utilizados en los diferentes componentes del sistema, y se menciona la inversión que realizó la empresa comparada contra lo que estaba a punto de perder.

## Capítulo 1. Introducción

Desde su aparición sobre la tierra, el hombre sintió la necesidad de protegerse. Desde las primeras sociedades humanas una de las principales funciones del Estado fue administrar justicia y proveer seguridad.

A medida que las sociedades evolucionaron, las causas que generan la inseguridad se hicieron más complejas. Por lo general, el Estado ha sido ineficiente para combatirlas, atacando antes a las consecuencias que a las causas que las provocan. Es por eso que la gente empezó a buscar protección según sus necesidades: mientras que para una casa en el campo de personas de bajos recursos o en un lugar despoblado un perro puede ser suficiente, para otras casas una alarma puede ser insuficiente.

La evolución de la ciencia y de la tecnología ha permitido evolucionar también a los sistemas de seguridad y de control de acceso de los que la sociedad puede disponer. En un principio consistían en medios físicos, tales como cerraduras, rejas, etc., fácilmente vulnerables con tiempo y herramientas; animales tales como perros guardianes; vigilantes tales como policías, serenos, veladores. Todos estos sistemas fueron evolucionando hasta crear sistemas más complejos como cerraduras electrónicas, alarmas o sistemas de observación por video.

La mayoría de las ocasiones la expectativa es la protección de los accesos a personas no autorizadas. Esta necesidad va más allá de los hogares y se encuentra en oficinas, fábricas, laboratorios, almacenes, escuelas, etc.

En la actualidad son muchos los sistemas de control de acceso electrónicos que aparecen en el mercado para solventar la demanda existente. Se pueden encontrar una multitud de lectores, tarjetas, y programas informáticos que lo realizan, pero realmente la diferencia está en el cumplimiento de las expectativas de los clientes.

Un sistema de control de acceso como tal es fácil de conseguir: un teclado, una cerradura eléctrica y solo accederán aquellas personas que conozcan la clave. Pero esto ya no es suficiente, puesto que al paso del tiempo la clave es conocida y lo que en un principio era un control de acceso termina convirtiéndose en un simple mecanismo para abrir puertas.



Una de las soluciones en la actualidad es la identificación de las personas mediante su huella digital. Gracias a su unicidad y constancia a través del tiempo, las huellas digitales han sido utilizadas para la identificación de las personas por más de un siglo, más recientemente volviéndose automatizada debido a los avances en las capacidades de la computación.

El uso práctico de huellas digitales como método de identificación de individuos se ha utilizado desde finales del siglo XIX cuando Sir Francis Galton<sup>1</sup> definió algunos de los puntos o características a partir de las cuales las huellas digitales podían ser identificadas.

La identificación por huella digital comienza su transición a la automatización a finales de los años 60 junto con la aparición de las tecnologías de computación. En 1969 hubo un empuje mayor por parte del Buró Federal de Investigaciones (FBI) para desarrollar un sistema para automatizar sus procesos de identificación por huellas digitales, el cual rápidamente se había vuelto abrumador y requería de muchas horas hombre para realizarlo manualmente.

En 1975 el FBI fundó el desarrollo de escáneres de huella digital y la tecnología de extracción de minucias<sup>2</sup>, lo cual condujo al desarrollo de un lector prototipo. Este trabajo condujo al desarrollo del primer algoritmo de reconocimiento de huellas digitales funcional, utilizado en el FBI para estrechar la búsqueda de humanos.

La tecnología de huellas digitales continuó mejorando y para el año 1981 varios sistemas estatales en los Estados Unidos y otros países habían implementado sus propios sistemas autónomos, desarrollados por diferentes proveedores. Durante esta evolución, la comunicación y el intercambio de información entre sistemas fueron pasados por alto, originando que una huella digital recogida con un sistema no podía ser buscada en otro. Esto llevó a la necesidad de desarrollar estándares de procesamiento y reconocimiento de huellas digitales.

---

<sup>1</sup> Primo del científico Sir Charles Darwin, centró su interés en el estudio de las diferencias individuales de las capacidades humanas. Identificó patrones comunes en las huellas digitales y desarrolló un sistema para clasificarlas que aun se utiliza en nuestros días.

<sup>2</sup> Las minucias son las pequeñas imperfecciones que los humanos tienen en las huellas digitales y son las que diferencian una huella digital de otra.

Así fue como a finales del siglo XIX comenzaron a aparecer los productos comerciales de verificación de huellas digitales para controles de acceso, para autenticación<sup>1</sup>, y para verificación<sup>2</sup>. Ahora se cuenta con tecnología que permite administrar varios miles de huellas digitales y verificar la identidad de las personas en base a las mismas.

En este escrito se describe un sistema para control de acceso utilizando la tecnología de huella digital que se desarrolló para una empresa llamada “House of Fuller” ubicada cerca del centro de Xochimilco. Se trata de una empresa holandesa trasnacional cuyo giro es la fabricación de cosméticos, cremas, lociones, y artículos varios para el hogar. La empresa vende sus productos con la ayuda de catálogos en lo que se conoce como “venta directa” o “venta por catálogo”.

Se trata de una empresa que cuenta con dos instalaciones: una planta de producción donde se fabrican, envasan y empaquetan los productos, y una instalación de oficinas administrativas donde también tienen almacenes. Tiene aproximadamente 1600 empleados para los cuales se requieren controlar las faltas y retardos, las entradas al comedor, y la apertura de puertas de acceso en ciertas áreas restringidas en las cuales solo se permite el paso a los empleados autorizados.

La necesidad de la empresa era controlar los aspectos antes mencionados utilizando la tecnología de huella digital. Para esto la empresa adquirió algunas terminales con lector de huella digital integrado y es precisamente la implementación del sistema lo que se describe en este trabajo.

El sistema que se diseñó está conformado básicamente de cuatro componentes: algunas terminales con lector de código de barras y de huella digital, un servidor de base de datos y un servicio Web que funciona como puente para que las terminales se comuniquen con la base de datos y un módulo de reportes en las computadoras de quienes quedarán encargados de administrar el sistema. Todo esto utilizando tecnología .NET por varios motivos que más adelante se explicarán.

---

<sup>1</sup> Se le llama “autenticar” al proceso de conocer la identidad de una persona. También se le conoce como “identificar”.

<sup>2</sup> Se le llama “verificar” al proceso de comprobar que una persona realmente es quien se dice ser.

## Capítulo 2. Descripción del Problema

El problema que se pretende solucionar con este proyecto es la administración y el control de acceso de los empleados a las diferentes áreas de la empresa. Algunas de estas áreas son: la fábrica, el comedor, y la puerta de entrada a las instalaciones generales de la misma.

Actualmente la empresa cuenta con algunas terminales con lector de código de barras integrado y sistema operativo de línea de comandos, las cuales se busca reemplazar por terminales capaces de trabajar con huella digital. Las nuevas terminales por las que se reemplazarán se colocarán en varios lugares, sumando un total de nueve terminales que trabajarán de forma casi idéntica pero con ligeras diferencias, dependiendo del tipo de proceso que realizará cada una (reloj checador, control de acceso al comedor, etc.).

La empresa adquirió las nueve terminales, pero para su mala fortuna estas terminales tenían ciertas limitaciones que en un principio no fueron contempladas, provocando con esto el problema que se describe en los siguientes párrafos.

Las huellas digitales se almacenan en módulos de huella digital que vienen integrados dentro de las terminales, y están ligadas a un número de empleado, también conocido como “Id de empleado”, de forma tal que el módulo puede saber si una cierta huella digital corresponde con un número específico de empleado.

Existen dos formas de realizar el reconocimiento de las huellas digitales que se conocen como “*uno a uno*” y “*uno a muchos*”:

La forma “*uno a uno*” consiste en que cuando el empleado coloca su dedo en el lector de huellas, debe proporcionar también su número de empleado. El módulo de huellas digitales compara la huella proporcionada contra la huella que se tiene almacenada relacionada con ese número de empleado, y solo hay dos opciones: corresponde o no corresponde.

En el caso de “*uno a muchos*” el empleado solamente coloca su dedo en el lector. Entonces el módulo de huellas debe comparar esa huella contra todas y cada una de las que tiene almacenadas, devolviendo el número del empleado correspondiente en caso de que encuentre correspondencia con alguno, o un

valor de no correspondencia si la huella no se encuentra almacenada en el módulo.

La forma de reconocimiento “uno a uno” es mucho más veloz pues solo compara la huella proporcionada en ese momento contra una de las que ya están almacenadas. A diferencia del reconocimiento “uno a muchos” que debe compararla contra todas las huellas almacenadas, que pueden ser miles, y decidir si corresponde con alguna.

Ambos modos de trabajo tienen sus ventajas y sus desventajas. Uno es más veloz y además soporta el almacenamiento de más huellas digitales, pero de alguna forma se debe introducir un número de empleado para realizar el reconocimiento. El otro modo solo necesita la huella del empleado, pero está más limitado en cuanto a la capacidad de almacenamiento y al tiempo de respuesta.

En la tabla 2.1 se muestran algunos ejemplos de capacidades de almacenamiento de algunos equipos que se pueden encontrar en el mercado.

<b>Módulo de Huella Digital</b>	<b>Capacidad de Almacenamiento de Huellas Digitales</b>	
	<b>Modo 1:1</b>	<b>Modo 1:N</b>
MV1250 de Bioscrypt	4000	No soportado
FIM10 de BioEnable	200	200
SFM3500 de Suprema	9000	9000

Tabla 2.1 *Capacidad de almacenamiento de algunos módulos de huella digital*

Las terminales que adquirió la empresa poseen una capacidad de almacenar y reconocer hasta 4000 huellas en el modo “*uno a uno*”, pero no soportan el modo “*uno a muchos*”.

Este es precisamente el problema que se debe solucionar: los equipos deben soportar las huellas digitales de 1600 empleados en el modo “uno a muchos”. Este dato puede fluctuar porque, por ejemplo para la temporada navideña la empresa contrata varias decenas de personas para trabajar solo uno o dos meses. Además de que los fabricantes recomiendan almacenar dos huellas por cada empleado debido a que las huellas digitales por naturaleza pueden alterarse un poco debido a factores externos como el clima o el uso frecuente de detergentes, o incluso puede ser que alguien se corte y durante algunos días requiera de un vendaje en el dedo. Esto arroja un total aproximado de 3200 huellas que deben ser almacenadas en el módulo.

El motivo por el que la empresa adquirió los equipos con tales características fue porque inicialmente el proceso de autenticar a los empleados se diseñó de la siguiente forma: El empleado deslizaría su credencial en un lector de código de barras y posteriormente colocaría su dedo en el lector de huellas digitales. Para lograr esto las terminales que la empresa adquirió cumplían perfectamente con lo solicitado, pues se trata de un reconocimiento de tipo “*uno a uno*” y para esto las terminales soportan, como ya se mencionó, hasta 4000 huellas.

Posteriormente los directores administrativos de la empresa solicitaron que el personal se autenticara utilizando solamente su huella digital, sin tener que deslizar ninguna credencial.

Para esto, la empresa buscó una solución, pensando incluso en la posibilidad de revender las terminales y adquirir otras que sí soportaran la capacidad de procesamiento requerida. Es aquí donde inició mi participación en el desarrollo del sistema.

Por lo demás no hay mayor problema puesto que la empresa cuenta con la infraestructura de red necesaria para la comunicación de los equipos, varios servidores con diferentes sistemas operativos y diferentes manejadores de bases de datos, así como del capital necesario para realizar la inversión. Aunque, claro está, se debía buscar la solución más eficiente y a su vez la más económica posible.

## Capítulo 3. Posibles Soluciones

Se encontraron tres alternativas factibles que solucionan el requerimiento de la empresa, y se explican en las siguientes líneas. Se incluye también una posible cuarta alternativa con la explicación de porque no es factible.

### Opción 1:

Varias empresas venden lectores de huella digital que se conectan a la computadora mediante un puerto serie universal (también conocido por sus siglas en inglés como puerto USB). Tienen un aspecto similar a un ratón de computadora aunque son un poco más pequeños. Estos lectores básicamente obtienen una fotografía de la huella digital de la persona y la almacenan ya sea en formato de mapa de bits (también llamado BMP) o en formato JPG<sup>1</sup> en un directorio específico. Además estas empresas venden un SDK<sup>2</sup> para trabajar con las imágenes de huella digital obtenidas.

Los SDK's son un conjunto de bibliotecas de programación generalmente disponibles para Lenguaje C, Visual Basic y Java. Entre otras funcionalidades proporcionan rutinas para relacionar cada huella digital con un número de empleado, y rutinas para comparar las huellas almacenadas contra una huella recién obtenida, para así saber si se trata de un empleado registrado en el sistema.

La desventaja de esta alternativa es que se necesitaría una computadora para cada lector de huella digital que se requiera colocar. Esto conlleva a colocar también un teclado, un ratón, un monitor y un equipo de respaldo de corriente eléctrica para cada una. Si se toma en cuenta que el ambiente de trabajo de estos equipos será principalmente en fábrica y que en algunos sitios donde deben ser colocados no se cuenta con el espacio suficiente para poner una computadora, entonces esta opción deja de ser viable. Por ejemplo, en la entrada de uno de los comedores solo hay una pared.

---

<sup>1</sup> Formato de imagen creado por un grupo de fotógrafos. Se le llama así por ser las siglas en ingles de "Join Photographic Group".

<sup>2</sup> SDK es una abreviación de Software Development Kit. Se refiere a un conjunto de bibliotecas de programación creadas por el fabricante de un equipo de hardware, que pueden utilizarse con cualquier lenguaje de programación y mediante las cuales es posible controlar dicho hardware.

## Opción 2:

Existen en el mercado algunas terminales inteligentes fueron diseñadas para este tipo de situaciones. Se trata de equipos pequeños, algunos casi tan pequeños como una computadora de bolsillo y otros un poco más grandes, pero sin superar los 20 x 20 cm. de tamaño. Tienen un aspecto similar al que se muestra en la figura 3.1.



Figura 3.1 *Terminales inteligentes con módulo de huella digital. De izquierda a derecha: BioAccess v2 de BioEnable, S3K LAN de Biometrix y NAC3000 de BioEnable. Tomadas de las páginas Web de los fabricantes*

Son varias las empresas que fabrican este tipo de aparatos, con sistema operativo de consola o con sistema operativo Windows. Otros ejemplos son la “M4 Terminal” de Chinazhiwen® o la “MR650” de Unitech®. Fueron equipos de este tipo los que adquirió la empresa.

Estos equipos cuentan con batería de respaldo, por lo cual no necesitan forzosamente un equipo externo que suministre energía en caso de existir un corte en la alimentación, sino que basta con un regulador. Además tienen un pequeño teclado integrado, o por lo menos algunos botones, y tienen una pantalla de cristal líquido y algunas incluso tienen una pantalla táctil. Con esto eliminan la necesidad de colocar dispositivos de entrada como el teclado y el ratón, o de visualización como el monitor. De hecho el equipo como tal es una pequeña computadora.

Aunque estos equipos no pueden trabajar con los lectores de huella digital que se conectan en el puerto serie universal por motivos de incompatibilidad del sistema operativo que estos últimos requieren, realmente no lo necesitan, pues algunas terminales ya tienen integrado un módulo lector de huellas digitales. Claro está que esto limita más las opciones porque, de los pocos modelos existentes en el mercado, son menos aún los que tienen integrado dicho módulo.

En esta opción lo que se haría es comprar otras terminales, probablemente de la misma marca y similar apariencia, que sí soporten las poco mas de 3200 huellas digitales en el modo de reconocimiento “*uno a muchos*”. La desventaja de esta opción es que se requiere una inversión de la misma magnitud de la que ya se realizó, además de dedicarse a revender o deshacerse de alguna forma de las terminales que ya se adquirieron.

### Opción 3

Aunque las terminales que adquirió la empresa ya tienen integrado un módulo lector de huella digital, éstas se pueden abrir y el modulo se puede cambiar por otro que sí soporte lo solicitado: más de 3200 huellas. Esto no es nada fácil pues todos los módulos son diferentes y cada uno tiene su propio conjunto de instrucciones para controlarlo.

La apariencia de estos módulos se muestra en la figura 3.2. Son pocas las empresas que los venden, sin embargo en este sentido no hay mucho que buscar puesto que para poder reemplazarlo, el nuevo sensor debe tener dimensiones idénticas o por lo menos muy similares a las del módulo que traen instalado de fábrica.

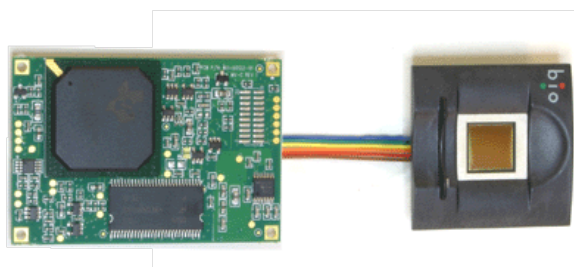


Figura 3.2 Módulo de huella digital MV1250 de Bioscrypt®. Tomada de la página Web del fabricante

Aquí cabe hacer una aclaración. El módulo de huella digital se compone de dos dispositivos: un módulo pequeño que captura la imagen de la huella, similar a una cámara fotográfica, y otro módulo más grande que realiza el proceso de reconocimiento. En esta solución lo que se requiere cambiar es el módulo de reconocimiento y dependiendo del tipo de cables y/o de señales que éste maneje puede ser o no necesario reemplazar también el módulo que captura la imagen de la huella.



Afortunadamente varios módulos de diferentes empresas son del mismo tamaño o casi del mismo tamaño que aquel que viene instalado de fábrica, haciendo referencia tanto al módulo de procesamiento como al módulo que captura la imagen de la huella.

Además junto con estos dispositivos el fabricante también vende un SDK, disponible generalmente para Lenguaje C, Visual Basic y Java. En este caso se tendrían que utilizar las funcionalidades de las rutinas del SDK para controlar directamente al módulo lector de huellas.

Así que la opción consiste en reemplazar el módulo de huellas digitales que tienen integrados las terminales por uno similar, pero que soporte almacenar más de 3200 huellas en el modo “*uno a muchos*”.

Esta opción es, por mucho, la más viable puesto que es realizable en un tiempo razonable y no es muy costosa: cada módulo lector de huellas digitales cuesta aproximadamente la quinta parte de lo que cuesta la Terminal completa.

#### **Opción 4:**

Una cuarta opción sería utilizar el conjunto de comandos que controlan directamente el módulo lector de huellas digitales que ya tienen integrado las lectoras que adquirió la empresa para obtener la imagen de cada huella (ya sea en formato BMP o JPG), almacenar estos archivos en la memoria de cada Terminal y, posteriormente, cuando un empleado coloque su huella en el lector, realizar el reconocimiento mediante algún algoritmo que nosotros mismos implementemos.

Esta opción tiene una desventaja: los algoritmos para el reconocimiento de las huellas digitales no están documentados, solo se explican a grandes rasgos, y la empresa no esta dispuesta a invertir en el desarrollo de un algoritmo de reconocimiento debido a que es un área que aún se encuentra en el estado del arte, y el desarrollo de un nuevo algoritmo de ese tipo puede demorar varios años. Además es algo ya existente en el mercado y es más fácil y económico adquirir y utilizar un dispositivo que ya realice el reconocimiento requerido en vez de tratar de “reinventar la rueda”.

## Capítulo 4. Justificación de la Solución

La solución que se eligió para sacar adelante el desarrollo fue la tercera, en la que se adquirirán nuevos módulos lectores de huella digital con la capacidad de proceso y velocidad necesarios para manipular las huellas de los más de 1600 empleados que tiene la empresa.

El dispositivo elegido fue el SFM3500 de la marca Suprema®, pues además de ser el más rápido y confiable actualmente en el mercado y de cumplir con los requerimientos necesarios, debido a su tamaño encaja perfectamente en el espacio destinado dentro de la Terminal para tal efecto.

La tabla 4.1 muestra un comparativo del tiempo que tardan diferentes módulos en realizar un reconocimiento de tipo “*uno a muchos*” dependiendo de la cantidad de huellas que tengan almacenadas.

Módulo de Huella Digital	Cantidad de Huellas	Velocidad de Reconocimiento
MV1250 de Bioscrypt	No soporta reconocimiento “uno a muchos”	
FIM10 de BioEnable	200	1.2 seg.
SFM3500 de Suprema	1000	680 mseg.

Tabla 4.1 *Velocidad de procesamiento de algunos módulos de huella digital*

Se trata entonces de adquirir tantos módulos como Terminales se requieran instalar, y con estos reemplazar los módulos preinstalados dentro de cada Terminal, y se deberá realizar la programación del módulo utilizando el SDK que proporciona el fabricante.

Para realizar el desarrollo se eligió la plataforma .NET debido a que para comunicarse con el servidor, la Terminal cuenta con un puerto de red, de forma tal que se puede conectar a la Intranet<sup>1</sup> de la empresa. Esto permite comunicarse mediante Servicios Web<sup>2</sup>. Aunque los Servicios Web se pueden implementar prácticamente con cualquier infraestructura, por ejemplo Windows, Linux, MacOS, etc., y con cualquier lenguaje de desarrollo como .NET, Java, PHP, etc., .NET proporciona gran versatilidad y facilidad para trabajar con los mismos. Además de que es una de las plataformas con las

---

<sup>1</sup> Se le llama Intranet a la red local de una empresa.

<sup>2</sup> Un Servicio Web es un servicio que contiene rutinas de programación, que se aloja en un servidor Web y se accede a sus rutinas mediante el protocolo HTTP como si se tratara de una página Web.

cuales las políticas de la empresa permiten desarrollar. Para este desarrollo la empresa solo ofreció dos opciones: .NET o Java.

Quedando solo dos posibilidades, una de ellas se elimina casi automáticamente: las terminales que se adquirieron tienen sistema operativo Windows CE, que es parte de la plataforma .NET. De forma tal que la aplicación no requerirá de instalar otros *frameworks*<sup>1</sup> o máquinas virtuales, sino que bastará con compilar la aplicación y copiar el archivo ejecutable en la Terminal para que este pueda trabajar.

De otra forma, se hubiese tenido que instalar la máquina virtual de Java en su edición para equipos móviles cada vez que la Terminal tuviera que reestablecerse. Esto es debido a que la memoria principal de las Terminales es de tipo RAM y por tal motivo se pierde cualquier información al ser reestablecidas o al agotarse la batería.

Estas terminales también cuentan con una pequeña memoria de tipo Flash, pero por lo mismo de ser pequeña no es buena idea desperdiciarla con este tipo de software, sino que mas bien es preferible aprovecharla, por ejemplo, con una base de datos compacta donde se almacenen las entradas y salidas del personal o con las fotografías que la Terminal capture mientras esperan su turno para ser enviadas al servidor.

Este problema no ocurre con .NET, puesto que el framework sobre el que trabaja ya viene instalado en la memoria ROM de las Terminales.

El lenguaje seleccionado fue Visual Basic .NET. Esto es por ser un lenguaje que, además de ser parte de la familia de lenguajes con los que se puede desarrollar para el sistema operativo Windows CE, es uno de los lenguajes más exitosos del mundo debido a la rapidez con que permite crear las aplicaciones.

Como manejador de bases de datos dentro de las terminales se seleccionó SQL Server Mobile. La elección se derivó del mismo motivo: ejecutando sobre el sistema operativo Windows CE no se requiere instalar ningún componente extra de terceros.

---

<sup>1</sup> Se le llama “framework” o “marco de trabajo” a la arquitectura de software sobre la que trabajan las aplicaciones. Puede estar compuesto por el sistema operativo, el motor de base de datos, y cualquier otro elemento que necesiten las aplicaciones para ejecutarse.

El manejador de base de datos que se utilizó en el servidor es SQL Server debido a que se utilizó una base de datos de SQL Server ya existente para otro sistema más grande por que, una vez terminado el desarrollo, éste terminará siendo un módulo más de ese sistema.

Acerca del Servicio Web se puede mencionar que, una vez elegido .NET como plataforma de desarrollo y Visual Basic .NET como lenguaje de desarrollo, también éste se creará con el mismo lenguaje de programación. La decisión como tal de utilizar un Servicio Web es debido a que son igualmente seguros pero más sencillos de utilizar que otras tecnologías como los *sockets*<sup>1</sup>. Además, no es necesario adquirir herramientas de terceros para el envío de información entre las Terminales y el servidor.

Finalmente, para el módulo de reportes se utilizará Visual FoxPro debido a que, como ya se mencionó, éste módulo quedará integrado dentro de otro sistema más grande ya existente, el cual está construido con Visual FoxPro.

Exactamente lo mismo ocurre con el módulo desde el cual se configurará cada Terminal. Este módulo se utilizará para indicarle a cada una el tipo de proceso que debe realizar: reloj checador, control de comedor, etc., o los parámetros bajo los cuales debe trabajar. Este módulo también será parte del sistema ya existente.

---

<sup>1</sup> Los sockets son una forma en que dos programas se pueden comunicar aún cuando se encuentren en diferente computadora.

## Capítulo 5. Implementación

En este capítulo se explicará detalladamente el desarrollo realizado tanto en el hardware como en el software de las terminales y del servidor. También se explican algunos aspectos importantes como el diagrama general del sistema, la estructura de la base de datos y el servicio Web.

### 5.1 Funcionamiento General de la Aplicación

En la figura 5.1 se muestra el diagrama a bloques del sistema completo. Aquí puede observarse la interacción de los diferentes elementos: las terminales, el servicio Web del servidor, la base de datos, etc. Partiendo de este diagrama, se explicará primero el funcionamiento general del sistema y posteriormente se detallará cada módulo que lo compone.

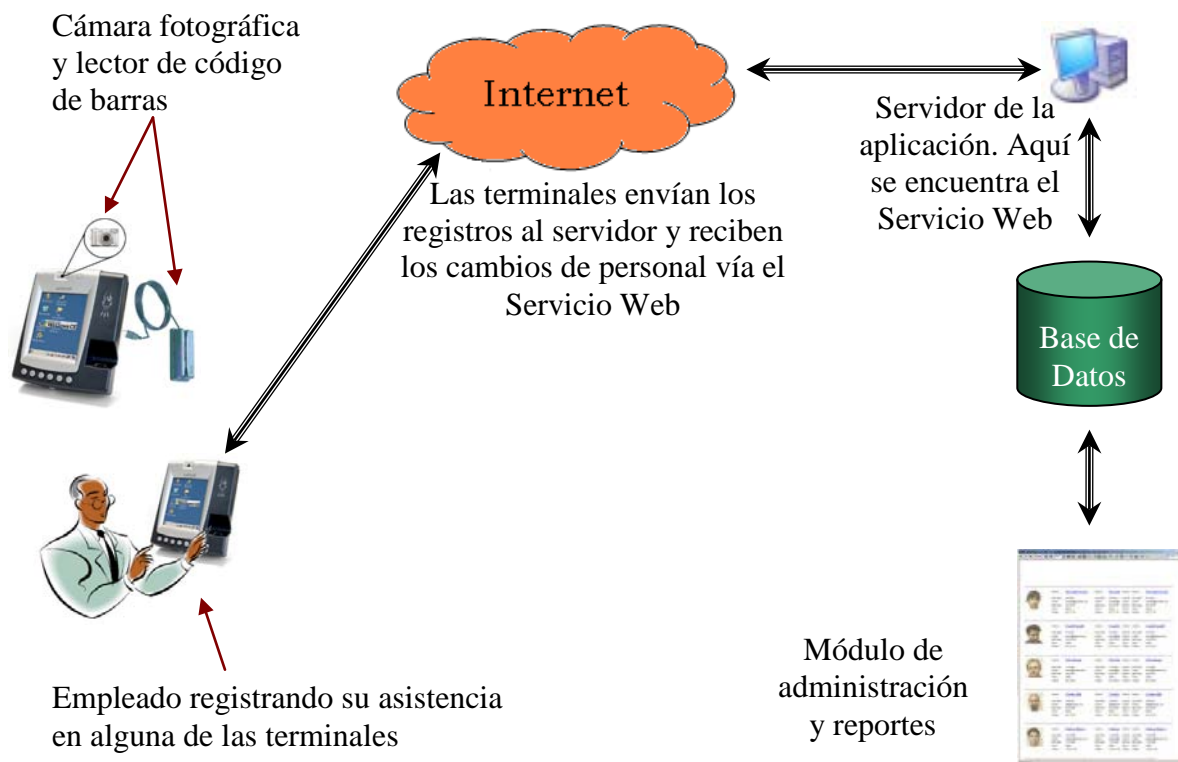


Figura 5.1 *Diagrama general del sistema*

El diseño del sistema consta de varias terminales con lector de código de barras y de huella digital que representan la interfaz con el usuario. Se colocará una Terminal en cada puerta de acceso que se requiera controlar.

Cuando el empleado necesite pasar por una puerta controlada tiene que colocar su dedo sobre el lector de huellas digitales. La Terminal realizará la verificación correspondiente y procederá enseguida a permitir o denegar el paso mediante mensajes en pantalla, sonidos y/o activación de relevadores, dependiendo del resultado obtenido.

Algunas terminales además de autenticar y decidir si se tiene acceso o no, también se encargarán de abrir una puerta por medio de una cerradura eléctrica. Esto se realizará así porque existen algunas áreas dentro de la empresa a las que solamente algunos empleados pueden tener acceso.

Las terminales almacenarán en una base de datos interna todos los movimientos que realice cada empleado. En general se trata de tres tipos de movimientos:

- Registro de hora de entrada y salida. En este caso la Terminal hace las veces de un reloj checador.
- Registro de accesos al comedor. Este movimiento permite conocer cuantas veces y a que hora entró al comedor cada empleado.
- Registro de apertura de puerta. Permite saber quién abrió una puerta, cuantas veces y a que hora lo hizo. Esto es útil para controlar a aquellos empleados que salen de su área principal de trabajo y tardan mucho tiempo en regresar, o aquellos que salen constantemente sin justificación alguna.

Aunque la empresa no lo solicitó, también se podrían almacenar los movimientos que fueron rechazados, esto es, aquellos empleados que traten de entrar a un área prohibida para ellos, o aquellas personas que traten de ingresar al comedor sin ser empleados de la empresa.

Por ejemplo, se podría capturar y guardar la huella digital de un empleado que trató de acceder a un área restringida. Además, las terminales tienen integrada una cámara fotográfica con la que se le podría tomar una fotografía al intruso.

Las terminales requieren alguna forma de enviar todos los movimientos que almacenan a un servidor de base de datos. Para esto se utilizará un Servicio Web: las terminales se conectarán a la red e invocarán un Servicio Web que reciba los movimientos realizados en cada una y los grabe en la base de datos del sistema.

Finalmente, el administrador del sistema podrá obtener, desde su propia computadora, los reportes necesarios acerca de la entrada y salida del personal por los diferentes accesos de la empresa.

## 5.2 La Base de Datos del Sistema

Ya se comentó anteriormente la razón por la cual se eligió SQL Server como manejador de base de datos para el sistema. Ahora se explicará detalladamente como se conforma la base de datos. La figura 5.2 muestra el diagrama de las tablas que la componen.

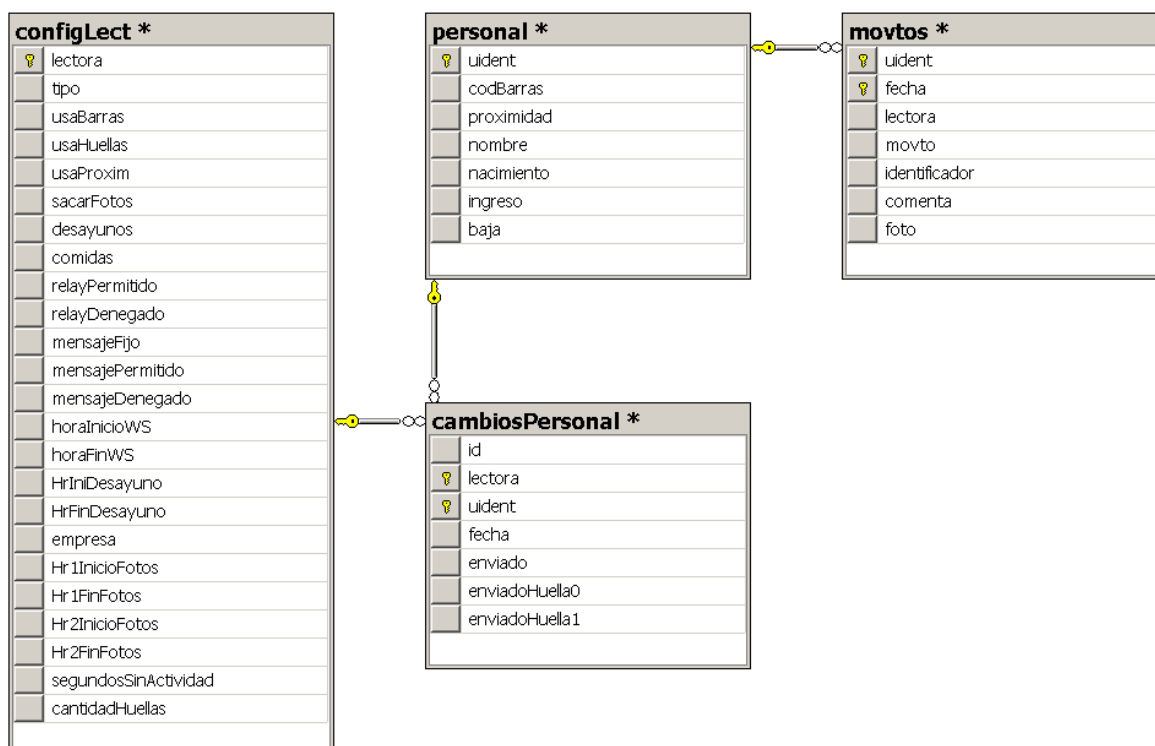


Figura 5.2 Base de datos del sistema

La tabla de *Personal* representa a todo el personal de la empresa que debe registrar sus entradas y/o salidas en cualquiera de las terminales. Esta tabla debe contener prácticamente a todos los empleados que laboran en la empresa, pues todos tienen derecho a entrar al comedor y también todos deben registrar su entrada y su salida en cualquiera de las puertas de acceso.

El campo *uident* es el número identificador del empleado. Los campos *codBarras* y *proximidad* corresponden al texto grabado en la credencial del empleado. Actualmente se utiliza credencial con código de barras, pero con

esto el sistema estará preparado para utilizar también credenciales de proximidad. Son estos tres campos los que permiten identificar a un empleado por medio de las tres tecnologías: huella digital (campo *uident*), credencial con código de barras (campo *codBarras*) y credencial de proximidad (campo *proximidad*).

Los campos *nombre* y *nacimiento* almacenan el nombre completo del empleado y su fecha de nacimiento respectivamente. La fecha de nacimiento se utilizará para saludar de forma especial al empleado el día de su cumpleaños. Los campos *ingreso* y *baja* almacenan la fecha en que el empleado se incorporó a la empresa y, cuando sea el caso, la fecha cuando el empleado fue dado de baja. Esto permite incrementar un poco la seguridad verificando que el personal ya haya iniciado a laborar en la empresa (y no que falten varios días para que inicie su contrato) y que no lo hayan despedido o no haya renunciado.

La tabla *ConfigLect* almacena los parámetros de configuración de cada Terminal. A continuación se describen sus campos:

Los campos *lectora* y *tipo* almacenan el nombre que tendrá la lectora (por ejemplo, “A” o “CC”) que es como se identificará a cada Terminal, y el tipo de trabajo que debe realizar (reloj checador, control de acceso al comedor, o control de acceso en áreas restringidas). De esta forma cualquier Terminal puede cambiar su modo de trabajo con solo realizar un pequeño cambio en este campo. Los campos *UsaBarras*, *UsaHuellas* y *UsaProxim* se utilizan para indicarle a la Terminal las tecnologías que debe usar para autenticar al personal. Esto permite, en un momento dado, desactivar el modulo de huellas digitales y autenticar solo mediante la credencial de código de barras, o activar ambos para que primero se tenga que deslizar la credencial y posteriormente colocar el dedo en el lector de huellas. La configuración que tendrán las Terminales en esta empresa será: activar solamente la autenticación por medio de la huella digital y desactivar las otras dos.

El campo *sacarFotos* es una bandera para indicar a la Terminal si debe capturar una fotografía en caso de una entrada denegada. Los campos *desayunos* y *comidas* solo tienen sentido cuando se trata de la Terminal que controlará el acceso al comedor. Estos campos indican la cantidad de desayunos y/o comidas a los que tiene derecho un empleado cada día.



Los campos *relayPermitido* y *relayDenegado* son banderas que indican a la Terminal si se debe activar un relevador en caso de un acceso permitido o denegado, respectivamente. Estos relevadores serán los encargados de abrir alguna cerradura eléctrica o de encender algún semáforo con luz verde o roja, o incluso de hacer sonar alguna sirena en señal de alarma.

Los campos llamados *mensajeFijo*, *mensajePermitido* y *mensajeDenegado* corresponden con los textos que aparecerán en las pantallas de las terminales. Son mensajes como: “Bienvenido, coloque su dedo en el lector de huella digital” (mensaje fijo), “Adelante” (mensaje permitido) o “Acceso Denegado” (mensaje denegado). Esto permite cambiar la apariencia de la pantalla de cada Terminal con solo actualizar estos tres campos.

Los campos *horaInicioWS* y *horaFinWS* le indican a las Terminales las horas de cada día en las que se pueden comunicar con el servidor (por ejemplo, de 6am a 10pm). Esto es porque la empresa a veces apaga el servidor en las noches. De igual forma, los campos *HrIniDesayuno* y *HrFinDesayuno* le indican a la Terminal que controla el acceso al comedor los horarios en que la comida se considera como desayuno (por ejemplo, de 6am a 9am). Los campos *Hr1InicioFotos*, *Hr1FinFotos*, *Hr2InicioFotos* y *Hr2FinFotos* le indican los horarios en los que la Terminal debe capturar fotografías en caso de que se haya activado la opción de sacar fotos.

El campo *empresa* es para uso interno del programa. Se utiliza para controlar los pequeños detalles que cada empresa solicite. Esto permite que el sistema sea general y se pueda vender en cualquier empresa, y no estar siempre atados a la primera empresa que lo compró. El campo *segundosSinActividad* le indica a las terminales el lapso de tiempo que debe transcurrir sin que haya actividad en la misma para iniciar el envío de información al servidor. Esto evitará que la Terminal intente enviar datos y/o archivos al servidor en las horas pico de registro. Finalmente el campo *cantidadHuellas* le indica a las terminales cuantas huellas se deben registrar de cada empleado (1 o 2).

La tabla *CambiosPersonal* se utiliza para controlar en envío de información relativa a los empleados entre las terminales y el servidor mediante el servicio Web. Los campos llave de esta tabla son el número de empleado (*uident*) y el nombre de cada lectora (*lectora*). Es de esta forma que cada lectora puede saber que empleados debe tener registrados en su base de datos interna. Por ejemplo, si la tabla contiene los registros de la tabla 5.1:

Terminal	Empleado
A	Juan
A	Pedro
B	Lolita
B	Cholita
B	Jaime
C	Juan
C	Lolita

Tabla 5.1 Ejemplos de datos almacenados en la tabla *CambiosPersonal*

Significa que en la Terminal “A” se registrarán los empleados “Juan” y “Pedro”, en la Terminal “B” se registrarán los empleados “Lolita”, “Cholita” y “Jaime”, y en la Terminal C se registrarán los empleados “Juan” y “Lolita”.

O, visto de otra forma, “Juan” se registrará en las terminales “A” y “C”, “Lolita” en las terminales “B” y “C”, “Pedro” solo en “A”, y “Cholita” y “Jaime” en “B”.

De esta forma, cada lectora sabrá de antemano que empleados tiene registrados. El campo *enviado* indica si ya se ha enviado el registro correspondiente de la tabla *personal*. Si tiene el valor “1” significa que ya se ha enviado, si tiene el valor “0” entonces aún no ha sido enviado.

Así, si se requiere agregar un empleado de nuevo ingreso en una Terminal, primero se deben agregar sus datos en la tabla *personal* y, posteriormente, se debe agregar un registro en la tabla *CambiosPersonal* con el valor “0” en el campo *enviado*. Al hacerlo, cuando la Terminal se comunique con el servidor mediante el servicio Web y le solicite el listado de los nuevos empleados, el servicio Web incluirá a este último. Posteriormente la Terminal solicitará los datos completos de cada uno de los nuevos empleados y, al hacerlo, el servicio Web cambiará el valor del campo *enviado* para asignarle un “1” y descartarlo así de posteriores listados de empleados nuevos.

De igual forma, cuando se requiera modificar los datos de algún empleado se debe establecer el valor del campo *enviado* en “0”, para que en la siguiente solicitud del listado de nuevos empleados (aunque no sea precisamente éste el caso) el número de empleado se incluya en la lista. Será responsabilidad de cada Terminal determinar si se trata de un registro nuevo que se debe agregar o de un cambio de un registro ya existente.

Los campos *EnviadoHuella0* y *EnviadoHuella1* son similares, solo que estos controlan no el envío de un registro de un empleado, sino el envío de un archivo de huella digital. De forma tal que cuando se capture la huella digital de un empleado en una Terminal, ésta lo envía inmediatamente al servidor y se asigna el valor “0” en el campo *EnviadoHuella0*. Posteriormente cuando otras terminales soliciten al servidor las huellas de los nuevos empleados o de aquellos que se hayan capturado de nuevo, por ejemplo, porque tuvieron un pequeño accidente y tienen un vendaje en el dedo con el que se registraban, éste devolverá los nuevos archivos de huellas digitales basándose en el valor del campo *EnviadoHuella0*, y una vez enviado cada archivo se le asignará el valor “1” para indicar que dicha huella digital ya ha sido enviada a la Terminal correspondiente.

El campo *EnviadoHuella1* se utiliza en caso de que se capturen dos huellas digitales de cada empleado.

Basándose en esto, si la tabla tiene los datos de la tabla 5.2, significa que cuando la Terminal “B” solicite las nuevas huellas digitales, el servidor le debe devolver la primera huella digital de “Cholita”. Cuando la Terminal “C” haga la misma solicitud, el servidor le debe devolver ambas huellas digitales de “Lolita”.

Lectora	Empleado	Enviado	EnviadoHuella0	EnviadoHuella1
A	Juan	1	1	1
A	Pedro	1	1	1
B	Lolita	1	1	1
B	Cholita	1	0	1
B	Jaime	1	1	1
C	Juan	1	1	1
C	Lolita	1	0	0

Tabla 5.2 Ejemplos de datos almacenados en la tabla *CambiosPersonal*

NOTA: Esto es solo demostrativo, pues el campo de *empleado (uident)* realmente almacena el número de empleado y no el nombre

El campo de *fecha* de la misma tabla solo se utiliza para ordenar los cambios en el personal o en sus huellas digitales, y el campo *ID* se utiliza para identificar cada registro de la tabla y funciona como llave cuando se requiere modificar cualquiera de los campos que controlan el envío de los datos.

La tabla *Movtos* almacena los movimientos registrados en cada Terminal, esto es, las entradas y salidas del personal, los accesos al comedor, la apertura de puertas, etc. Los campos de ésta tabla representan lo siguiente:

El campo *uident* es el número de empleado que realizó el movimiento. El campo *fecha* almacena la fecha y la hora del movimiento (a que hora entró, a que hora salió a comer, etc.). El campo *Lectora* es la Terminal que registró dicho movimiento (“A”, “B”, “C”, etc.). *Movto* es el tipo de movimiento realizado ((C)omedor, (P)uerta o (R)eloj checador).

El campo *identificador* indica el tipo de identificación que utilizó el empleado para autenticarse (credencial con (C)ódigo de barras, credencial de (P)roximidad o (H)uella digital). El campo *comenta* es un campo donde la Terminal puede construir algún mensaje preestablecido que indique algo especial acerca del movimiento. El campo *foto* almacena el nombre de algún archivo con formato JPG que corresponda con una fotografía capturada por la Terminal cuando se tiene activada la opción de tomar fotos.

Estas son todas las tablas de la base de datos principal del sistema. A partir de ellas es posible obtener prácticamente cualquier tipo de información como por ejemplo: “Que empleados llegaron tarde el mes pasado” o “Cuantos accesos al comedor intentó hacer Juan el día lunes”. Este tipo de consultas son las que se podrán realizar en el módulo de reportes.

### 5.3 El Servicio Web

El Servicio Web de este sistema es el puente de comunicación entre las terminales y el servidor o la base de datos. Es mediante éste que las terminales pueden manipular la información de la base de datos, o recibir y enviar archivos desde y hacia el servidor.

Cabe aclarar que todas las rutinas del servicio Web se invocan desde cada una de las terminales. A continuación se describe la función que realiza cada una de ellas:

**EnviarArchivoAlServidor:** Permite a las terminales enviar archivos de cualquier tipo al servidor, y es mediante esta rutina que se envían las fotografías tomadas por las terminales.

**RecibirArchivoDesdeElServidor:** Es la que ejecutan las terminales cuando requieren que el servidor les envíe algún archivo. Se utiliza para que el servidor les envíe a las terminales el logo de la empresa y los sonidos que se deben reproducir para acompañar a los mensajes de “Adelante” y “Acceso Denegado”.

**RecibirConfiguracionDesdeElServidor:** Mediante esta rutina el servidor envía a cada Terminal su configuración bajo la cual debe trabajar.

**RecibirCambiosPersonal:** Es la que utilizan las terminales para solicitar al servidor el listado de los empleados de nuevo ingreso o de aquellos a los que se les han modificado sus datos. El servidor regresa el listado de los empleados separados por comas.

**RecibirPersonalDesdeElServidor:** Una vez que la Terminal tiene en su poder el listado de los empleados de nuevo ingreso (o de aquellos a los que se les han modificado sus datos), entonces solicita, uno por uno, los nuevos datos de cada empleado por medio de ésta rutina.

**EnviarMovimientoAlServidor:** Esta rutina es la que utiliza cada Terminal para enviar al servidor, uno por uno, todos los movimientos que tiene registrados.

**RecibirCambiosHuellas:** Es similar a la rutina que obtiene el listado de los empleados de nuevo ingreso (o cuyos datos han cambiado), solo que ésta obtiene los empleados que se les ha capturado alguna huella digital. También regresa un listado separado por comas.

**RecibirHuellaDesdeElServidor:** Una vez obtenido el listado de los empleados que se les ha capturado su huella digital, la Terminal solicita mediante esta rutina, una por una, todas las huellas digitales que se encuentren en el servidor.

**EnviarHuellaAlServidor:** Cada vez que alguna Terminal graba una huella digital, esta es enviada al servidor para que otras terminales puedan solicitarla y grabarla cada una en su propio módulo de huellas digitales.

**RecibirArchivoConfigHuellaDigitalDesdeElServidor:** Esta rutina se utiliza para enviar un archivo especial que contiene información de configuración para los módulos de huella digital que están integrados en las Terminales.

Con este conjunto de rutinas es como las terminales obtienen toda la información que necesitan para trabajar, como configuración, archivos, registros de empleados, etc., y envían al mismo toda la información que recolectan en el transcurso del día, como movimientos, fotografías, huellas digitales, etc.

Algunas de estas rutinas devuelven una estructura con los datos obtenidos de una tabla de la base de datos, o reciben una estructura con los datos que deben grabar. Otras reciben o devuelven un archivo en forma de arreglo de bytes.

En la figura 5.3 se muestra un ejemplo de una estructura y como ésta corresponde de forma casi idéntica con una tabla de la base de datos:

```
Public Structure Personal
    Public ExitoError As Estatus
    Public errorDesc As String
    Public uident As Integer
    Public codBarras As String
    Public proximidad As String
    Public nombre As String
    Public nacimiento As Date
    Public ingreso As Date
    Public baja As Date
End Structure
```

personal *	
	uident
	codBarras
	proximidad
	nombre
	nacimiento
	ingreso
	baja

Figura 5.3 Correspondencia entre una tabla de la base de datos y una estructura definida en el programa

Nota: Todos los fragmentos de código que se muestran en este trabajo están codificados en el lenguaje de programación Visual Basic .NET

Como puede observarse, la estructura *Personal* es casi un espejo de la tabla *Personal*, con la diferencia de que agrega algunos campos para facilitar la manipulación de algún error eventual durante la ejecución de la aplicación.

En el siguiente seudo código se puede observar la forma en que la rutina correspondiente arma una estructura de tipo *Personal* a partir de los datos de la tabla *Personal*, para posteriormente devolverla como valor de retorno de la función.

```
<WebMethod()> _
Public Function RecibirPersonalDesdeElServidor(ByVal id As Long)
```

As Personal

```
<Iniciar bloque Try-Catch de control de errores>

<Leer datos de la persona con el id correspondiente en la base de
datos>

<Con los datos obtenidos, armar una estructura de tipo "Personal">
<Actualizar la tabla CambiosPersonal para indicar que ese registro
ya ha sido enviado>

<Terminar bloque Try-Catch de control de errores>

<Devolver la estructura como resultado de la función>
```

End Function

Ver Anexo A.1

En este caso lo primero que hace la rutina es consultar en la base de datos la información del empleado correspondiente, basándose para ello en el campo *Id*. Posteriormente, la rutina llena los campos de una estructura de tipo *Personal*. Finalmente, antes de devolver como respuesta dicha estructura, la rutina actualiza una bandera para indicar que dicho empleado ya fue enviado a la Terminal que lo solicitó.

Además, la estructura dispone de un campo para almacenar un estatus de éxito o de error, según sea el caso. Si en algún momento ocurriera cualquier tipo de error, entonces la rutina simplemente regresa el mensaje de error.

El siguiente pseudo código es otro ejemplo de las rutinas que posee el servicio Web, en la que la Terminal solicita al servidor una huella digital y éste la devuelve en forma de un arreglo de bytes:

```
<WebMethod()> _
Public Function RecibirHuellaDesdeElServidor(ByVal lectora As String,
                                             ByVal id As Integer, ByVal index As Byte) As Byte()

    <Iniciar bloque Try-Catch de control de errores>

    <Verificar si el archivo de huella digital existe para el número de
    empleado solicitado. Si no es así, generar una excepción>

    <Si el archivo existe, leer su contenido y almacenarlo en un
    arreglo de bytes>

    <Actualizar la tabla CambiosPersonal para indicar que la huella
    digital ha sido enviada>

    <Terminar bloque Try-Catch de control de errores>
```

```
<Devolver el arreglo de bytes como resultado de la función>
```

```
End Function
```

**Ver Anexo A.2**

En esta rutina primero se lee la información de un archivo de huella digital si es que éste existe y se almacena en un arreglo de bytes. Posteriormente se actualiza una bandera para indicar que la huella digital ya ha sido enviada a la Terminal que la solicitó. Finalmente devuelve el arreglo como resultado de la rutina. En caso de error la rutina devuelve un arreglo de un solo byte con el valor cero, lo cual para la Terminal significará que ocurrió algún error en el procesamiento de la petición. En caso de que la huella digital no exista en el servidor se enviará un arreglo con un solo byte de valor “1”, que la Terminal deberá interpretar como una huella inexistente.

El otro tipo de rutinas que tiene el Servicio Web es aquel en el que el servidor envía a la Terminal un listado separado por comas de los empleados cuyos registros o cuyas huellas digitales han sufrido algún cambio. A continuación se muestra un ejemplo de una de estas rutinas:

```
<WebMethod()> _  
Public Function RecibirCambiosPersonal(ByVal lectora As String)  
                                                                    As String  
  
    <Iniciar bloque Try-Catch de control de errores>  
  
    <Obtener de la tabla CambiosPersonal los ID's de los empleados de  
    nuevo ingreso, de los que se les ha modificado algún dato y de los  
    que han sido dados de baja>  
  
    <Armar una cadena de caracteres conteniendo todos los ID's de los  
    empleados separados por comas>  
  
    <Terminar bloque Try-Catch de control de errores>  
  
    <Devolver la cadena de caracteres como resultado de la función>  
  
End Function
```

**Ver Anexo A.3**

En esta rutina primero se realiza la consulta en la base de datos de todos aquellos empleados cuyos datos han sufrido cambios. Después con el conjunto de datos obtenidos se arma una cadena que contendrá los números que identifican a cada empleado separados por comas (por ejemplo: “3, 45, 27, 82”). Finalmente se devuelve dicha cadena. En caso de error la rutina devolverá una cadena vacía.



Todas las rutinas contenidas en el Servicio Web tienen la forma de alguno de los tres tipos que se han descrito en los párrafos anteriores. Así es como las terminales acceden a la base de datos y/o a los archivos almacenados en el servidor.

Como componente adicional dentro del mismo Servicio Web hay una clase que se encarga del acceso a la base de datos. Esta clase encapsula la mayor parte del código necesario para realizar consultas y ejecutar comandos en la base de datos. La principal utilidad de esta clase es facilitar dichas operaciones.

Por ejemplo, para realizar una consulta simple de la tabla *Personal* las instrucciones que se requieren son similares a las siguientes:

```
Dim connection As New SqlConnection
Dim adapter As New SqlDataAdapter
Dim dataSet As New DataSet

connection.ConnectionString = "...
connection.Open()

adapter.SelectCommand = New SqlCommand("select * from personal",
                                         connection)
adapter.Fill(dataSet)

connection.Close()
```

Todo este conjunto de instrucciones sería necesario en cualquier parte del programa en la que se requiera realizar una consulta a la base de datos. De forma similar, para ejecutar algún comando, por ejemplo eliminar un registro de una tabla, el conjunto de instrucciones sería similar al siguiente:

```
Dim connection As New SqlConnection
Dim adapter As New SqlDataAdapter
Dim dataSet As New DataSet

connection.ConnectionString = "...
connection.Open()

adapter.DeleteCommand = New SqlCommand("delete personal
                                         where uident = 1", connection)
adapter.DeleteCommand.ExecuteNonQuery()

connection.Close()
```

Como casi todas las rutinas del servicio Web se encargarán de manipular la base de datos será necesario repetir estos dos conjuntos de instrucciones en

varios lugares. Para evitar esto se diseñó una clase que se encargará de todos los accesos a la base de datos. Con ésta clase la consulta de los registros de la tabla personal tendrá la siguiente apariencia:

```
Dim bd As New AccesoDatosSQL  
bd.LeerRegistrosDataset("select * from personal")
```

La eliminación de algún registro de la misma tabla será similar a la siguiente:

```
Dim bd As New AccesoDatosSQL  
bd.EjecutarDelete("delete personal where uident = 1")
```

Como puede observarse, el código resulta mucho más limpio y compacto, además es mucho más fácil de leer y también de codificar. Aunque en estos ejemplos no se ha utilizado ningún control de errores, la clase está perfectamente preparada para avisar cerca de cualquier error que pudiera ocurrir internamente. También, si fuera necesario, la clase está preparada para usar transacciones de base de datos.

## 5.4 La Base de Datos de las Terminales

Para la base de datos de las terminales se utilizó SQL Server Mobile. La estructura de las tablas que la componen es casi idéntica a la de la base de datos principal del sistema, solo que en esta base de datos solo hay tres tablas: *ConfigLect*, *Personal* y *Movtos*. De hecho estas tres tablas prácticamente son un espejo de las tablas de la base de datos principal, de forma tal que cuando se usa el Servicio Web para intercambiar información con el servidor básicamente lo que se está haciendo es copiar los registros de las tablas de una base de datos a la otra, eliminando de las terminales aquellos registros que ya se han copiado.

Por lo demás no hay nada de especial en esta base de datos. Se puede acceder a ella mediante una consola especial para el sistema operativo Windows CE y desde ahí es posible ejecutar instrucciones de SQL. La aplicación que se ejecutará en la Terminal tendrá acceso a la base de datos, al igual que lo hace el Servicio Web con su base de datos respectiva, por medio de una cadena de conexión. En el caso de la aplicación móvil que accede a la base de datos móvil la cadena de conexión tiene la siguiente forma:

```
Data Source = \Flash Storage\Lectora\basedatos\lectora.sdf Password = "
```

## 5.5 El Hardware de las Terminales

Con respecto al hardware involucrado en el desarrollo de la aplicación se explicará primero la composición interna de las terminales. Los componentes principales de las terminales se muestran en las figuras 5.4 y 5.5.

En estas imágenes no es posible mostrar algunos de los componentes principales como son la cámara fotográfica, el micrófono o la bocina, por el motivo de que se encuentran en una segunda tarjeta anexa a la tarjeta madre que está colocada por debajo de esta última, de forma tal que para poder visualizarlas sería necesario desmontar la tarjeta madre y algunos otros componentes, lo cual de ninguna forma fue necesario hacer para lograr la implementación del nuevo módulo de huella digital.

De igual forma, en las imágenes se han señalado algunos componentes secundarios con los que de alguna forma fue necesario involucrarse al momento de hacer la implementación del módulo de huella digital.

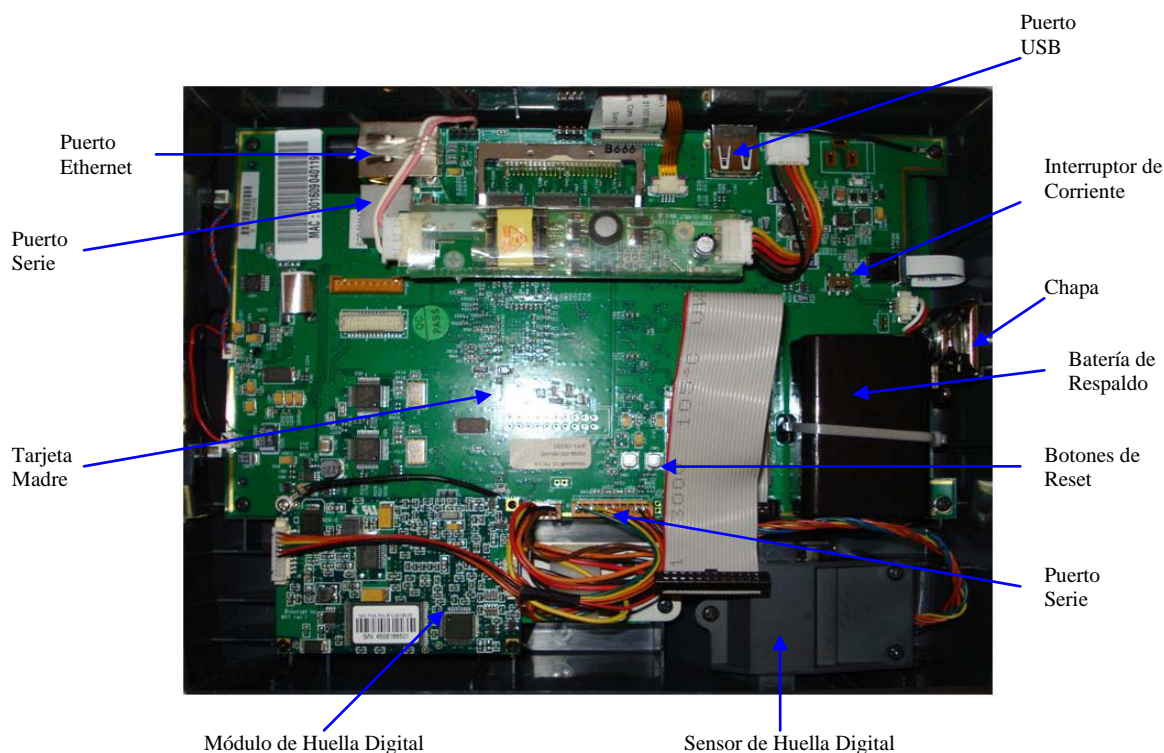


Figura 5.4 Componentes principales de las terminales

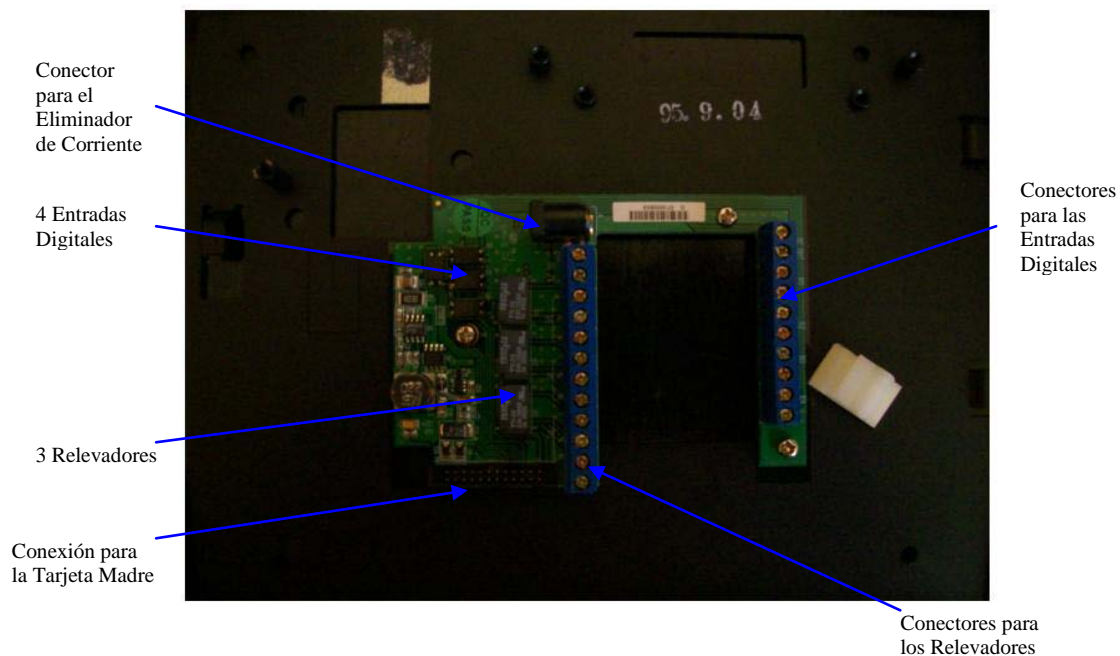


Figura 5.5 Componentes principales de las terminales

Una parte importante de este proyecto consistió en reemplazar el módulo de huella digital. Como se mencionó en la justificación de la solución elegida, el módulo original se reemplazaría por un módulo SFM3500 de la marca Suprema®. En la hoja de datos de dicho módulo se indica su tamaño exacto. En las figuras 5.6 y 5.7 se muestran diagramas relacionados con el tamaño del módulo y del sensor que lo acompaña. Partiendo de estos diagramas se pueden realizar mediciones en las terminales y decidir si el módulo y el sensor encajan bien en los espacios destinados para los mismos.

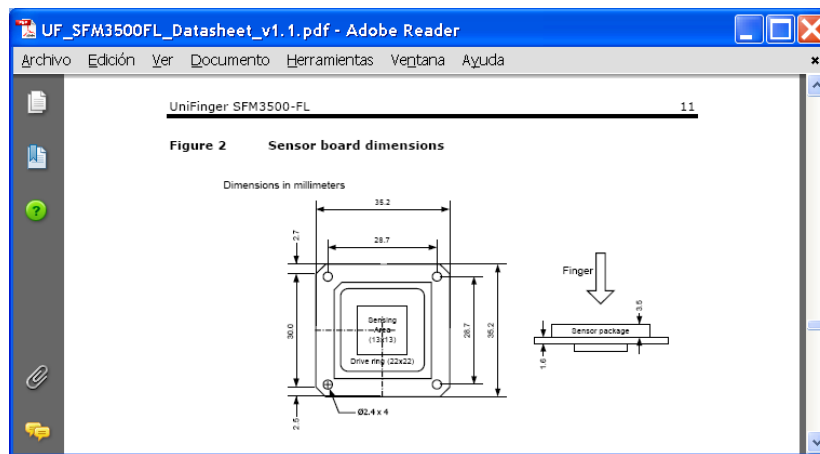


Figura 5.6 Diagrama de dimensiones del sensor de huella digital. Tomado del documento UF\_SFM3500FL\_Datasheet\_v1.1.pdf publicado por la empresa Suprema®

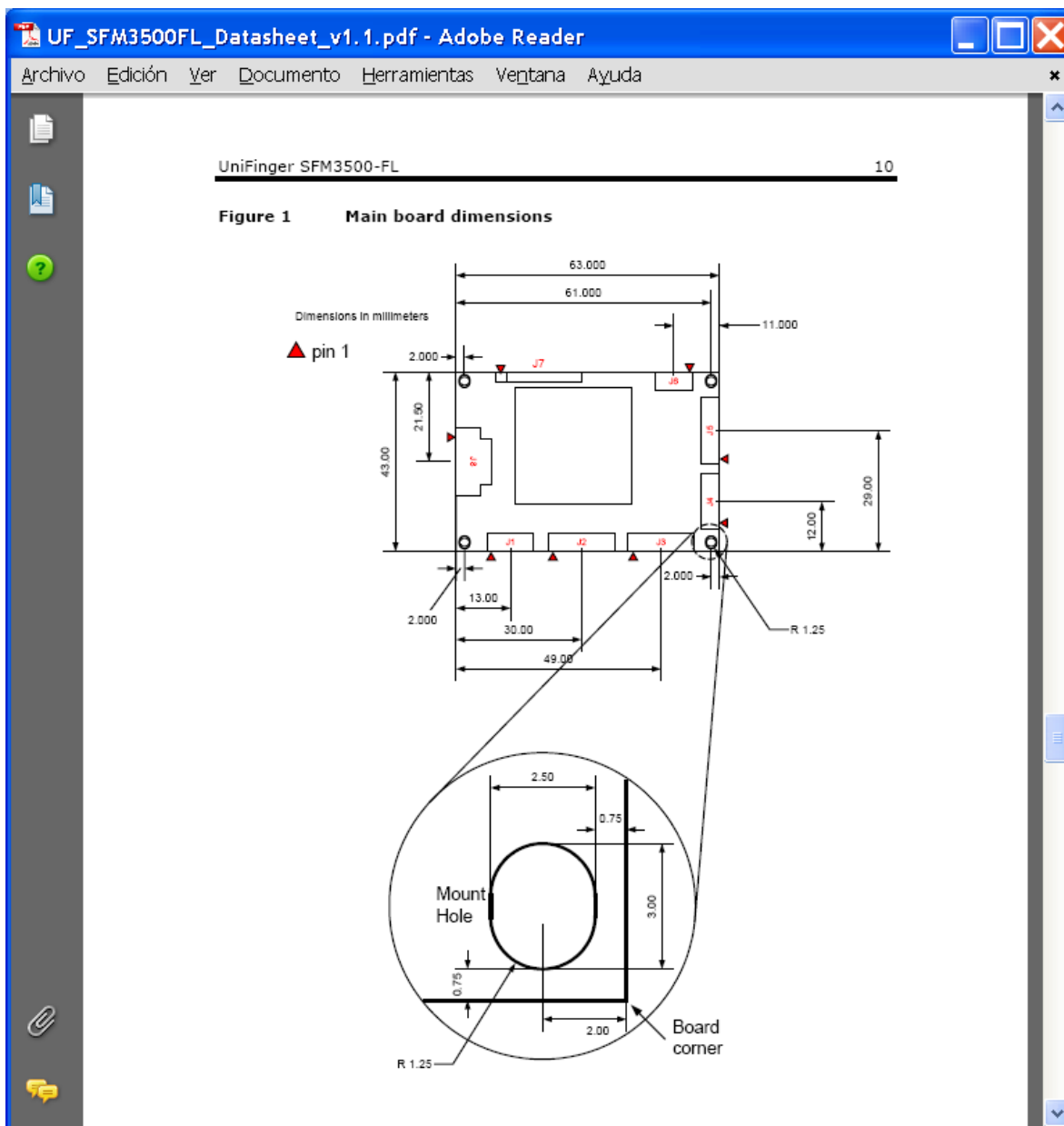
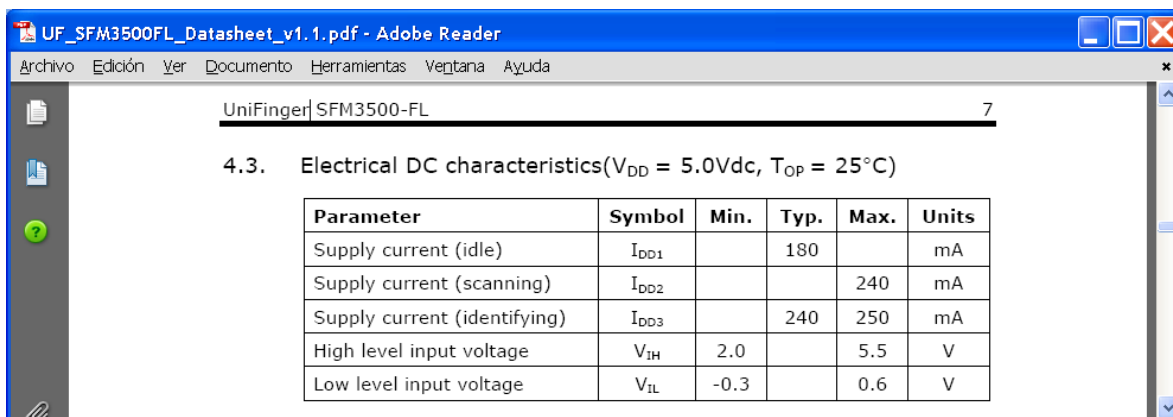


Figura 5.7 Diagrama de dimensiones del módulo de huella digital. Tomado del documento UF\_SFM3500FL\_Datasheet\_v1.1.pdf publicado por la empresa Suprema®

También en la hoja de datos del módulo se especifican las características de voltaje y corriente con las que se debe de alimentar. Con esto no hubo mayor problema, pues los voltajes que maneja y su consumo de corriente son perfectamente compatibles con los de la Terminal. La figura 5.8 muestra las características eléctricas de estos dispositivos:



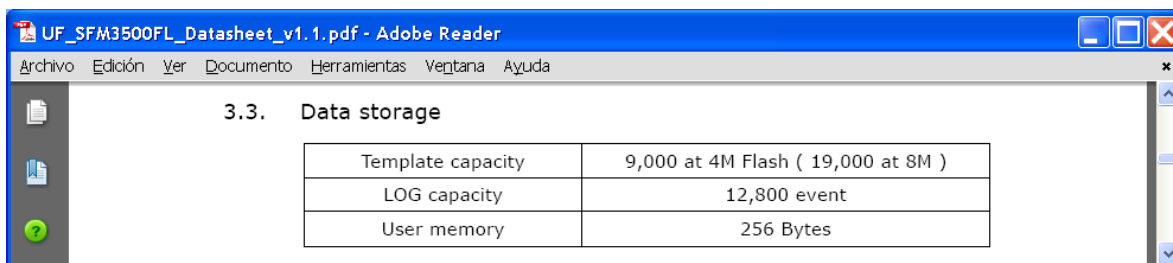
UniFinger|SFM3500-FL 7

4.3. Electrical DC characteristics( $V_{DD} = 5.0Vdc$ ,  $T_{OP} = 25^{\circ}C$ )

Parameter	Symbol	Min.	Typ.	Max.	Units
Supply current (idle)	$I_{DD1}$		180		mA
Supply current (scanning)	$I_{DD2}$			240	mA
Supply current (identifying)	$I_{DD3}$		240	250	mA
High level input voltage	$V_{IH}$	2.0		5.5	V
Low level input voltage	$V_{IL}$	-0.3		0.6	V

Figura 5.8 Características eléctricas del módulo de huella digital. Tomado del documento UF\_SFM3500FL\_Datasheet\_v1.1.pdf publicado por la empresa Suprema®

Finalmente, las últimas características con las que el módulo tenía que cumplir son las relacionadas con la capacidad de procesar las huellas digitales de 1600 empleados en el modo “uno a muchos”. En la figura 5.9 se demuestra que el módulo cumple con tales características:



3.3. Data storage

Template capacity	9,000 at 4M Flash ( 19,000 at 8M )
LOG capacity	12,800 event
User memory	256 Bytes

Figura 5.9 Características de almacenamiento del módulo de huella digital. Tomado del documento UF\_SFM3500FL\_Datasheet\_v1.1.pdf publicado por la empresa Suprema®

Una vez asegurado el hecho de que el módulo efectivamente funcionará adecuadamente dentro de la Terminal, el siguiente paso es la implementación del mismo.

El módulo de huella digital que viene montado originalmente en la Terminal se comunica con ésta por medio de un puerto serie, y de la misma forma lo hará el módulo con el cual se va a reemplazar. Así que lo primero que se necesita es conocer la relación de pines tanto del conector de puerto serie de la Terminal como del conector de puerto serie del módulo. Ambos datos se pueden obtener de la hoja de datos de cada dispositivo o en su defecto se pueden solicitar con el fabricante de los mismos. La relación de pines del puerto serie de la Terminal es la que se muestra en la tabla 5.3.

Número de Pin	Asignación
1	RS232 Transmisión de Datos (TX)
2	RS232 Recepción de Datos (RX)
3	NA
4	NA
5	VCC 5.0V
6	VCC 5.0V
7	NA
8	GND
9	NA
10	GND

Tabla 5.3 *Relación de pines del puerto serie de la terminal*

La relación de pines del puerto serie del módulo de huella digital es la que se muestra en la tabla 5.4:

Número de Pin	Asignación
1	GND
2	RS232 Transmisión de Datos (TX)
3	RS232 Recepción de Datos (RX)
4	NA
5	VCC 5.0V
6	NA
7	NA
8	NA
9	GND

Tabla 5.4 *Relación de pines del puerto serie del módulo*

Como puede deducirse de la relación de pines de ambos dispositivos, se debe de construir un cable con las características de la tabla 5.5:

No. de Pin en la Terminal	No. de Pin en Módulo de Huella Digital
1 (TX)	3 (RX)
2 (RX)	2 (TX)
5 (VCC)	5 (VCC)
8 (GND)	1 (GND)

Tabla 5.5 *Relación de pines para conectar el puerto serie de la Terminal con el puerto serie del módulo*



Esto con el fin de conectar correctamente tierra con tierra, voltaje con voltaje, y transmisor con receptor. Una vez montado el módulo en la Terminal y realizada esta conexión la Terminal tiene el aspecto de la figura 5.10.



Figura 5.10 Vista de la Terminal después de colocar el módulo SFM3500

Con respecto al lector de código de barras el único aspecto que cabe comentar es la conexión establecida mediante puerto serie entre el lector y la Terminal, y es que para realizar esta conexión se tuvieron que invertir dos de las señales del cable que las conecta. El motivo es que si estos cables se conectan tal cual están, esto es, el cable del lector de código de barras directamente con el cable que incluye la Terminal para conectarse en el puerto serie, las señales quedarían como se muestra en la tabla 5.6.

Esto es, transmisor con transmisor y receptor con receptor. Para solucionar esto bastó con agregar un pequeño trozo de cable en medio de ambos cuya única función sería cruzar las señales de comunicación.

En general, esto es todo lo relacionado con el hardware de la aplicación. Hay algunos detalles que se deben tomar en cuenta pero estos, además de ser



comunes prácticamente para cualquier equipo electrónico, están bien documentados en el manual de usuario de las terminales.

Asignación de Pin en la Terminal	Asignación de Pin en el Lector de Código de Barras
Tierra	Tierra
Transmisor (TX)	Transmisor (TX)
Receptor (RX)	Receptor (RX)

Tabla 5.6 *Relación de pines al momento de realizar la conexión si no se colocara un cable intermedio para cruzar las señales*

Por ejemplo, en el manual de usuario el fabricante recomienda conectar el eliminador de corriente durante una cierta cantidad mínima de horas de forma ininterrumpida para cargar totalmente la batería cuando el equipo se enciende por primera vez, o recomienda también que la pantalla táctil solamente se manipule utilizando la plumilla (también conocida como *stylus*) y no con los dedos o con otros objetos.

## 5.6 El Software de las Terminales

El software fue desarrollado con Visual Basic .NET el cual es un lenguaje visual orientado a objetos, lo que permitió crear algunas clases y objetos que correspondieran con algunos de los dispositivos de hardware involucrados en el desarrollo.

La ventaja de hacerlo así es que el programa resulta más intuitivo que con la programación estructurada. Esto a su vez se traduce en un fácil mantenimiento tanto para realizar correcciones como para hacer cualquier modificación que surja en el proceso de desarrollo.

El software de esta aplicación es muy grande así que se describirá por partes. Por un lado el programa se construyó utilizando varias clases que corresponden cada una con algún dispositivo de hardware como la cámara fotográfica o el módulo lector de huellas digitales; por otro lado se tienen clases que corresponden con elementos abstractos del sistema como el personal o los movimientos. También cuenta con algunas pantallas que conforman la interfaz gráfica con el usuario, y algunos módulos que almacenan constantes, variables y rutinas generales.

Se comenzará por un diagrama que muestra las diferentes clases que componen la aplicación y como interactúan entre ellas, y posteriormente se describirá como funciona cada una:

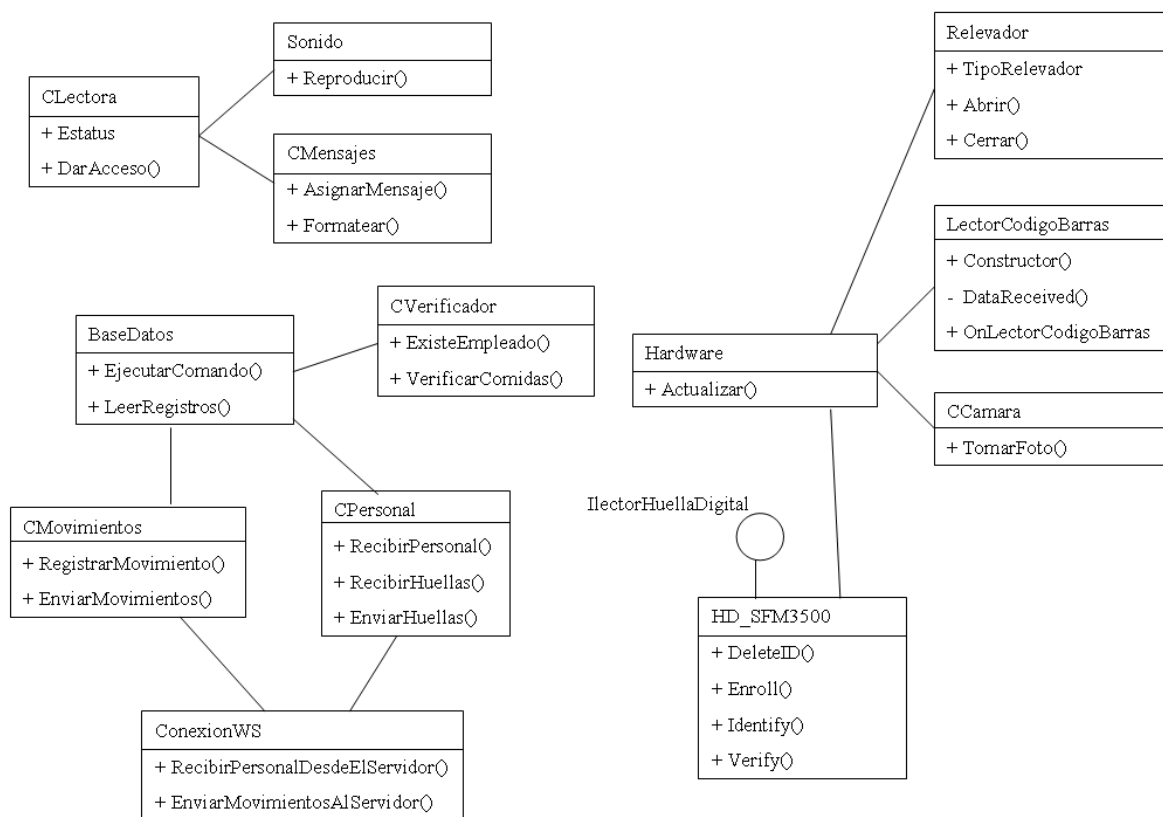


Figura 5.11 Diagrama de clases de la aplicación donde se muestran las propiedades, los métodos, y los eventos más importantes de cada una

### 5.6.1 La Clase que Manipula la Base de Datos: BaseDatos

Esta clase será la encargada de manipular la base de datos móvil que se encuentra en cada una de las Terminales. Será a través de ésta clase que se grabarán, modificarán y eliminarán los registros de la base de datos.

Internamente contiene los objetos nativos de .NET necesarios para realizar la conexión a la base de datos, y para ejecutar comandos y realizar consultas, además de un campo donde almacenará la descripción de cualquier error que ocurriera dentro de la clase.

En el siguiente código se muestra un ejemplo de cómo trabaja esta clase. En este caso se trata del método encargado de establecer una conexión con la base de datos de la Terminal:

```
' Conectar
Public Function Conectar() As Estatus

    errorDesc = ""
    Conectar = Estatus.EXITO

    Try

        conn = New SqlCeConnection(CONNECTION_STRING)
        conn.Open()
        command = conn.CreateCommand()
        adapter = New SqlCeDataAdapter()

    Catch ex As Exception

        errorDesc = ex.Message
        conn = Nothing
        Conectar = Estatus.ERROR

    End Try

End Function
```

Como puede observarse la clase está perfectamente preparada contra cualquier eventualidad que pudiera surgir. Encerrando todo el código importante dentro de un bloque de control de errores la clase se asegura de que siempre devolverá una respuesta confiable.

En esta rutina primero se crea el objeto *conexión*, posteriormente la conexión se abre y finalmente se crean dos objetos que se utilizarán durante todo el tiempo que se ejecute la aplicación. Si todo sale bien devuelve un estatus de “Éxito”. En caso de que ocurra algún error almacena una descripción del mismo y devuelve un estatus de “Error”.

Esta rutina utiliza una variable que almacena la cadena de conexión. En el caso de Windows CE la cadena de conexión se construye de la siguiente forma:

```
' Cadena de conexion
Public Shared CONNECTION_STRING As String = "Data Source = " &
                                             Arch.ARCH_BASEDATOS & "; Password = '"
```

Donde Arch.ARCH\_BASEDATOS es una constante que se encuentra dentro del módulo de constantes. La ventaja de utilizar constantes para este tipo de situaciones es que cuando se requiera modificar el valor que representan, basta con hacerlo solo una vez en un solo lugar.

Para ilustrar mejor éste funcionamiento se copiará un fragmento de código que define algunas constantes:

```
Module Arch

    Public Const ARCH_LOGO As String = ARCHIVOS_DIR & "logo.jpg"

    Public Const ARCH_BASEDATOS As String = BASEDATOS_DIR & "lectora.sdf"

    Public Const ARCH_SONIDO_BIEN As String = ARCHIVOS_DIR & "bien.wav"

    ...

End Module
```

Aquí puede verse cómo la base de datos es un archivo llamado “lectora.sdf”, o cómo el logotipo de la empresa es un archivo llamado “logo.jpg”. Sin embargo estas constantes usan, a su vez, otras constantes relacionadas con directorios. Se trata de directorios creados específicamente para la aplicación. A continuación se copiará también un fragmento de código donde se definen algunas de éstas constantes:

```
Module Dir

    Public Const APP_DIR As String = FLASH_STORAGE_DIR & "Lectora\"

    Public Const FOTOS_DIR As String = APP_DIR & "fotos\"

    Public Const ARCHIVOS_DIR As String = APP_DIR & "archivos\"

    Public Const BASEDATOS_DIR As String = APP_DIR & "basedatos\"

    Public Const HUELLAS_DIR As String = APP_DIR & "huellas\"

    ...

End Module
```

Aquí están las constantes que forman parte de los nombres de los archivos. En éste fragmento de código se puede observar cómo todos los directorios parten de uno mismo llamado APP\_DIR, esto es, se trata de un directorio que contiene a los subdirectorios “fotos”, “archivos”, “basedatos” y “huellas”.

Además el directorio APP\_DIR basa su nombre en otra constante. Se trata de una constante que corresponde con uno de los directorios principales del sistema operativo. A continuación se copiarán algunas definiciones para estos directorios:

```
Module Dir

    ...

    Public Const APPLICATION_DATA_DIR As String = "\Application Data\"

    Public Const MY_DOCUMENTS_DIR As String = "\My Documents\"

    Public Const FLASH_STORAGE_DIR As String = "\Flash Storage\"

End Module
```

De esta forma, el nombre completo archivo de base de datos se forma así:

```
\Flash Storage\Lectora\basedatos\lectora.sdf
```

El simple hecho de utilizar constantes para todos los directorios y archivos del sistema significa que si se requiere cambiar el nombre de algún directorio o cambiar su ubicación, basta con modificar el valor de la constante correspondiente y de forma automática el resto del sistema estará consciente del cambio.

Volviendo a la clase de *BaseDatos*, los otros dos métodos importantes que ésta posee son *EjecutarComando* y *LeerRegistrosDataset*. El primero se encarga de ejecutar instrucciones de tipo *insert*, *update* y *delete*, mientras que el segundo se encarga de realizar consultas. A continuación se explica lo que internamente hace cada una de estas rutinas:

```
' Ejecuta un comando
Public Function EjecutarComando() As Estatus

    errorDesc = ""
    EjecutarComando = Estatus.EXITO

    Try

        command.CommandText = query
        command.ExecuteNonQuery()

    Catch ex As Exception

        errorDesc = ex.Message
```

```
EjecutarComando = Estatus.ERROR

End Try

End Function
```

La rutina anterior recibe la instrucción que debe ejecutar mediante la variable miembro *query* y después simplemente la ejecuta. Al igual que todas las demás, devuelve un estatus de “Éxito” o de “Error”, así como una descripción del error en caso de que ocurra alguno.

```
' Hace un select
Public Function LeerRegistrosDataset() As Estatus

    errorDesc = ""
    LeerRegistrosDataset = Estatus.EXITO

    Try

        command.CommandText = query
        adapter.SelectCommand = command
        dataSet = New DataSet()
        adapter.Fill(dataSet)

    Catch ex As Exception

        errorDesc = ex.Message
        LeerRegistrosDataset = Estatus.ERROR

    End Try

End Function
```

Esta rutina primero recibe el comando SQL que utilizará para hacer la consulta (también mediante la variable miembro *query*), después crea un objeto de tipo *DataSet* que será donde recibirá los datos obtenidos, y finalmente ejecuta la consulta. El objeto *dataSet* es un miembro de la clase, así que una vez concluida la consulta es posible acceder a él mediante una instrucción similar a la siguiente:

```
miBaseDatos.dataSet...
```

Así es como todos los objetos y en general todas las partes del programa que lo necesiten acceden a la base de datos: mediante la clase *BaseDatos*. Con esta clase el programa resulta más sencillo e intuitivo tanto para codificar el resto de la aplicación como para revisar el código ya escrito, sabiendo de antemano

que ésta clase realiza su función como una caja negra (no importa lo que haya dentro, solo importa que hace lo que se requiere).

De esta forma, cuando se necesite acceder a la base de datos por cualquier motivo, las instrucciones que se ejecutarán serán similares a la siguiente:

```
bd.query = "select count(*) from personal where uident = " & uident  
  
If bd.LeerRegistrosDataset() = Estatus.ERROR Then  
  
Throw New Exception(bd.errorDesc)  
  
End If
```

### 5.6.2 La Clase que Manipula el Sonido: Sonido

Es la encargada de reproducir grabaciones de audio. En el sistema se utiliza para reproducir sonidos después de una validación de acceso, y tiene dos tipos de sonido: uno para indicar acceso permitido y otro para indicar acceso denegado.

La rutina principal de ésta clase se muestra a continuación:

```
Public Sub Reproducir(ByVal archivo As String)  
    Const SND_ASYNC = &H1  
    Const SND_FILENAME = &H20000  
  
    If IO.File.Exists(archivo) Then  
        WCE_PlaySound(archivo, IntPtr.Zero, SND_ASYNC Or  
                                                                SND_FILENAME)  
    End If  
End Sub
```

Se trata de una rutina nativa de Windows CE. Recibe como parámetros el nombre del archivo que se debe reproducir y la forma de reproducción: síncrona o asíncrona. Esta instrucción se encuentra dentro de una dll del sistema operativo, y su declaración dentro del programa tiene la siguiente forma:

```
Declare Function WCE_PlaySound Lib "\windows\CoreDll.dll" Alias  
"PlaySound" (ByVal szSound As String, ByVal hMod As IntPtr, ByVal flags  
As Integer) As Integer
```

Además, la clase sonido posee dos métodos para reproducir un sonido. Tales métodos se muestran a continuación:

```

Public Sub Bien()
    Reproducir(Arch.ARCH_SONIDO_BIEN_RAM)
End Sub

Public Sub Mal()
    Reproducir(Arch.ARCH_SONIDO_MAL_RAM)
End Sub

```

A su vez, estos métodos utilizan constantes, facilitando así la labor en caso de que se necesite cambiar el nombre de algunos de los archivos o su ubicación.

El hecho de tener ésta clase con los métodos antes descritos permite escribir instrucciones sencillas como la siguiente:

```
sonidos.Bien()
```

Haciendo más legible el código, que resultará más fácil tanto de leer como de codificar.

### 5.6.3 La Clase que Manipula la Cámara Fotográfica: CCamara

Esta clase controla la cámara fotográfica que tienen integradas las terminales. Esta cámara posee la capacidad de capturar video, pero en este sistema solamente se utilizará como cámara fotográfica, pues el espacio para almacenar archivos es muy limitado, y además los archivos de imágenes que se hayan capturado deben enviarse al servidor, proceso en el cual el envío de videos podría saturar el ancho de banda disponible para la comunicación.

La clase *CCamara* tiene un método para activar la cámara fotográfica y otro para desactivarla. Estos métodos tienen la finalidad de mantener encendido el hardware solamente cuando la configuración de la Terminal indique que se deben tomar fotografías, apagándolo en caso contrario.

Para controlar esto, ambos métodos utilizan rutinas de una dll que viene precargada en las terminales. Las declaraciones de las rutinas dentro del programa tienen la siguiente forma:

```

Declare Function OpenCamera Lib "\windows\CameraDll.dll" () As Boolean
Declare Sub ReleaseCamera Lib "\windows\CameraDll.dll" ()

```



Se usan como se muestra a continuación:

```
Public Sub Activar()  
  
    OpenCamera()  
    estatusHardware = Activo  
  
End Sub  
  
Public Sub Desactivar()  
  
    ReleaseCamera()  
    estatusHardware = Inactivo  
  
End Sub
```

Como puede notarse, la clase posee un indicador para saber si ese hardware está activo o no. Varias de las clases que corresponden con un dispositivo de hardware poseen el mismo indicador.

Una vez que se ha activado la funcionalidad de manipular la cámara fotográfica, el proceso de capturar una fotografía consta de tres pasos: Iniciar la vista previa del objetivo que se requiere fotografiar, capturar la fotografía, y detener la vista previa del objetivo.

El motivo por el cual se debe iniciar y detener la vista previa cada vez que se va a tomar una fotografía es porque para que la cámara pueda tomar la foto sólo puede hacerlo si se dispone de una vista previa del objetivo que se desea capturar, pero esa vista previa implica dibujar en la pantalla la imagen que la cámara está viendo. Esto es, la pantalla de la Terminal tiene que ser obstruida por la imagen que ve la cámara. Es por ello que no se puede dejar siempre iniciada la vista previa, sino que, por el contrario, se debe iniciar y detener cada que se requiera tomar una fotografía.

Las rutinas que inician y detienen la vista previa también se encuentran dentro de la misma dll y tienen la siguiente forma:

```
Declare Sub StartPreview Lib "\windows\CameraDll.dll" (ByVal left As  
Short, ByVal top As Short, ByVal width As Short, ByVal height As Short)  
  
Declare Sub StopPreview Lib "\windows\CameraDll.dll" ()
```

Los parámetros de la primera de ellas representan las coordenadas de un rectángulo dentro de la pantalla donde aparecerá la imagen del objetivo que se va a capturar.

Finalmente, para tomar la fotografía la rutina es la que se muestra a continuación:

```
Public Sub TomarFoto(ByVal uident As Integer,  
                    Optional ByVal x As Short = 10,  
                    Optional ByVal y As Short = 10,  
                    Optional ByVal ancho As Short = 100,  
                    Optional ByVal largo As Short = 100)  
  
    <Verificar de acuerdo a la tabla ConfigLect, si la hora es  
    apropiada para tomar fotografías>  
  
    <Iniciar bloque Try-Catch de control de errores>  
  
    <Generar el nombre de archivo a partir de la fecha, la hora, y el  
    número del empleado>  
  
    <Tomar la fotografía mediante las rutinas de la API>  
  
    <Terminar bloque Try-Catch de control de errores>  
  
End Sub
```

**Ver Anexo B.1**

Esta rutina primero determina si la configuración de la Terminal indica que se deben tomar fotos y se cerciora de que la cámara fotográfica esté activada. Después toma en consideración los horarios en los que, de acuerdo a la configuración, se deben tomar fotografías. Finalmente ejecuta los tres pasos antes mencionados para capturar la foto. El nombre del archivo lo construye en base al número del empleado y a la fecha y hora del sistema.

Si no se hubiese creado ésta clase, el conjunto de instrucciones necesarias para tomar una fotografía sería similar al siguiente:

```
archivo = FechaHora_Texto(Now, FormatoFechas.aaaammdd, "\", "\", "_")  
                                & "_" & uident & ".jpg"  
  
StartPreview(x, y, ancho, largo)  
  
CaptureImage(Dir.FOTOS_DIR & archivo, ancho, largo)  
  
StopPreview()
```

Todo esto sin tomar en cuenta aún los horarios en los que la Terminal debe sacar fotografías. Pero gracias a que se creó esta clase, el código se simplifica de la siguiente forma:

```
camara.TomarFoto(0)
```

### 5.6.4 La Clase que Manipula el Relevador: Relevador

La clase *Relevador* se encarga de controlar el estado de los relevadores de la Terminal, abriéndolos y cerrándolos de acuerdo a lo que el sistema necesite en un momento dado.

Contiene una variable miembro llamada *tipo*, que establece el tipo de relevador que representará cada instancia de la clase. En la aplicación solo existirán dos relevadores: relevador de acceso permitido y relevador de acceso denegado. Para esto la clase se basa en la siguiente enumeración:

```
' Tipo de Relevador
Public Enum TipoRelevador As Short
    Permitido
    Denegado
End Enum
```

El tipo de relevador que tendrá cada instancia de la clase lo recibirá como parámetro en el constructor de la misma, de la siguiente forma:

```
Public relPermitir As New Relevador(Relevador.TipoRelevador.Permitido)
Public relDenegar As New Relevador(Relevador.TipoRelevador.Denegado)
```

Así es como se crean los dos objetos *Relevador* que representarán a los dos relevadores de la Terminal que utilizará el sistema.

Los métodos más importantes de ésta clase son *Abrir* y *Cerrar* que se encargan, como su nombre lo indica, de abrir y cerrar cada relevador. A continuación se muestra la codificación de ambos métodos:

```
Public Sub Cerrar(ByRef inicio As frmInicio)

    If estatusHardware = Activo Then

        inicio.TimerRelays.Enabled = False
        SetRelay(rel, True)
        estatus = Cerrado
        inicio.TimerRelays.Enabled = True

    End If

End Sub

Public Sub Abrir()

    If estatusHardware = Activo Then
```

```
        SetRelay(rel, False)
        estatus = Abierto

    End If

End Sub
```

Dentro del método *Cerrar*, primero se cerciora de que el relevador se encuentre abierto, pues no tiene sentido cerrar un relevador que ya está cerrado. Después se detiene un temporizador que se encargará de abrir de nuevo el relevador algunos segundos más tarde. Posteriormente, se establece el estado del relevador mediante la instrucción *SetRelay*, cuya declaración es la siguiente:

```
Declare Function SetRelay Lib "\windows\SysIOApi.dll" (ByVal rel As Short, ByVal [On] As Boolean) As Boolean
```

Finalmente se asigna el valor a una bandera interna de la clase que indica que el relevador se encuentra cerrado y se habilita el temporizador encargado de volver a abrirlo.

El método *Abrir* es un poco mas sencillo. En éste solamente se establece el nuevo estado del relevador también mediante la instrucción *SetRelay*, y se asigna a la bandera el valor que indica que se encuentra abierto.

Ambos métodos utilizan dos constantes cuyo único propósito es ayudar a que el código resulte más legible. Se trata de las constantes *Abierto* y *Cerrado*:

```
Public Const Cerrado As Byte = 0
Public Const Abierto As Byte = 1
```

Una vez codificada ésta clase, las instrucciones en las que se utilizará tendrán la siguiente forma:

```
relPermitir.Abrir()
```

### 5.6.5 La Clase que Invoca las Rutinas del Servicio Web: ConexionWS

El objetivo de esta clase es encapsular y realizar toda la comunicación relacionada con el Servicio Web del sistema. Así, cualquier objeto o subrutina que requiera comunicarse con el Servicio Web tanto para enviar o recibir información como para enviar o recibir archivos lo hará por medio de ésta

clase. Por lo tanto, esta clase posee un método por cada método que ofrezca el Servicio Web. Así, por ejemplo, la clase *ConexionWS* contiene el método que se muestra a continuación, el cual se encarga de enviar una huella digital al servidor:

```
Public Function EnviarHuellaAlServidor(ByVal id As Integer,
                                       ByVal index As Byte) As Estatus

    <Iniciar bloque Try-Catch de control de errores>

    <Obtener los datos del archivo de huella digital y los almacena en
    un arreglo de bytes >

    <Realizar el envío mediante una instancia del servicio web>

    <Terminar bloque Try-Catch de control de errores>

    <Devuelve Éxito ó Error como resultado de la función >

End Function
```

**Ver Anexo B.2**

Este método solamente recibe el número de empleado del cual se debe enviar su huella digital y cuál de las dos posibles huellas que éste tenga grabadas se trata. La clase se encarga de localizar y leer el archivo correspondiente para enviarlo como un arreglo de bytes.

Es dentro de esta clase donde se crea el objeto correspondiente al Servicio Web, y de hecho se trata de un miembro de la clase:

```
Public Class ConexionWS

    Private ws As New Service

    ...

End Class
```

Como puede observarse, este método ejecuta una rutina del servicio Web, precisamente aquella que envía una huella digital al servidor, y regresa como resultado un estatus de éxito o error.

Lo mismo ocurre con todos los demás métodos de esta clase: ejecutan una rutina del Servicio Web y regresan un estatus de éxito o error, según sea el caso. A continuación se muestra otro método de la misma clase que se utiliza para enviar un movimiento al servidor:

```

Public Function EnviarMovimientosAlServidor() As Estatus

    <Iniciar bloque Try-Catch de control de errores>

    <Armar una estructura de tipo "Movimiento" con los datos del
    movimiento que se va a enviar>

    <Realizar el envío mediante una instancia del servicio web>

    <Terminar bloque Try-Catch de control de errores>

    <Devuelve Éxito ó Error como resultado de la función >

End Function

```

Ver Anexo B.3

En este caso los datos que se deben enviar como cuerpo del movimiento ya fueron obtenidos con otra clase que se explicará más adelante, y fueron dejados en el *dataSet* miembro de la clase *BaseDatos*. Así que aquí solamente se obtienen los valores que se encuentran dentro de dicho *dataSet* y se asignan a los campos de la estructura *Movimiento*. Una vez armada la estructura que se debe enviar, se envía por medio de la subrutina *EnviarMovimientoAlServidor* del Servicio Web.

La ventaja de tener una clase que se encargue de la comunicación con el Servicio Web es que cuando se tenga que hacer alguna adecuación al mismo, la mayor parte del código que se tendrá que corregir será el que se encuentra dentro de esta clase, reduciendo al mínimo las modificaciones necesarias en otras partes de la aplicación.

En general, la idea fundamental de ésta clase es que los objetos que requieran enviar información al servidor se limiten a obtener toda la información que se deba enviar y dejarla lista para que la clase *ConexionWS* la envíe. Entonces ésta clase tomará todos los datos que se tengan que enviar e invoca la rutina del Servicio Web correspondiente.

De igual forma, cuando un objeto requiera obtener información desde el servidor, solo tiene que solicitárselo a la clase *ConexionWS* y ésta se encargará de hacer todo el trabajo sucio, dejando los datos obtenidos en un lugar accesible por el objeto que los solicitó.

### 5.6.6 La Clase que Manipula el Lector de Código de Barras: LectorCodigoBarras

Esta clase se encarga de leer el código de barras de las credenciales que se deslicen por el lector. La comunicación con éste se realiza por medio de un puerto serie y su función es encapsular la complejidad de la comunicación serial con el lector para utilizarla con métodos y eventos sencillos.

El miembro más importante de esta clase es un objeto de tipo *SerialPort*. La clase *SerialPort* es una clase nativa de .NET y se utiliza precisamente para manipular el puerto serie. Al igual que las rutinas de comunicación serial de otros dispositivos y/o de otros lenguajes de programación, la clase *SerialPort* se puede parametrizar para configurar la comunicación. En este caso se configura en el constructor de la clase *LectorCodigoBarras*. El código se muestra a continuación:

```
Sub New()  
  
    ' Create a new SerialPort object with default settings.  
    puerto = New SerialPort()  
  
    ' Parametros del Puerto Serie  
    puerto.PortName = "COM1"  
    puerto.BaudRate = 9600  
    puerto.Parity = Parity.None  
    puerto.DataBits = 8  
    puerto.StopBits = StopBits.One  
    puerto.Handshake = Handshake.None  
  
    ' Set the read/write sizes  
    puerto.ReadBufferSize = 100  
  
    ' Manejador del puerto  
    AddHandler puerto.DataReceived, AddressOf DataReceived  
  
End Sub
```

La configuración del puerto se basó en la configuración predeterminada del lector de código de barras. Se trata de una velocidad de 9600, sin paridad, con 8 bits de datos y 1 bit de parada. También se estableció un tamaño de buffer de lectura de 100 bytes.

La mayoría de estos datos fueron tomados de la hoja de datos del dispositivo. En la figura 5.12 se muestran algunos de ellos.

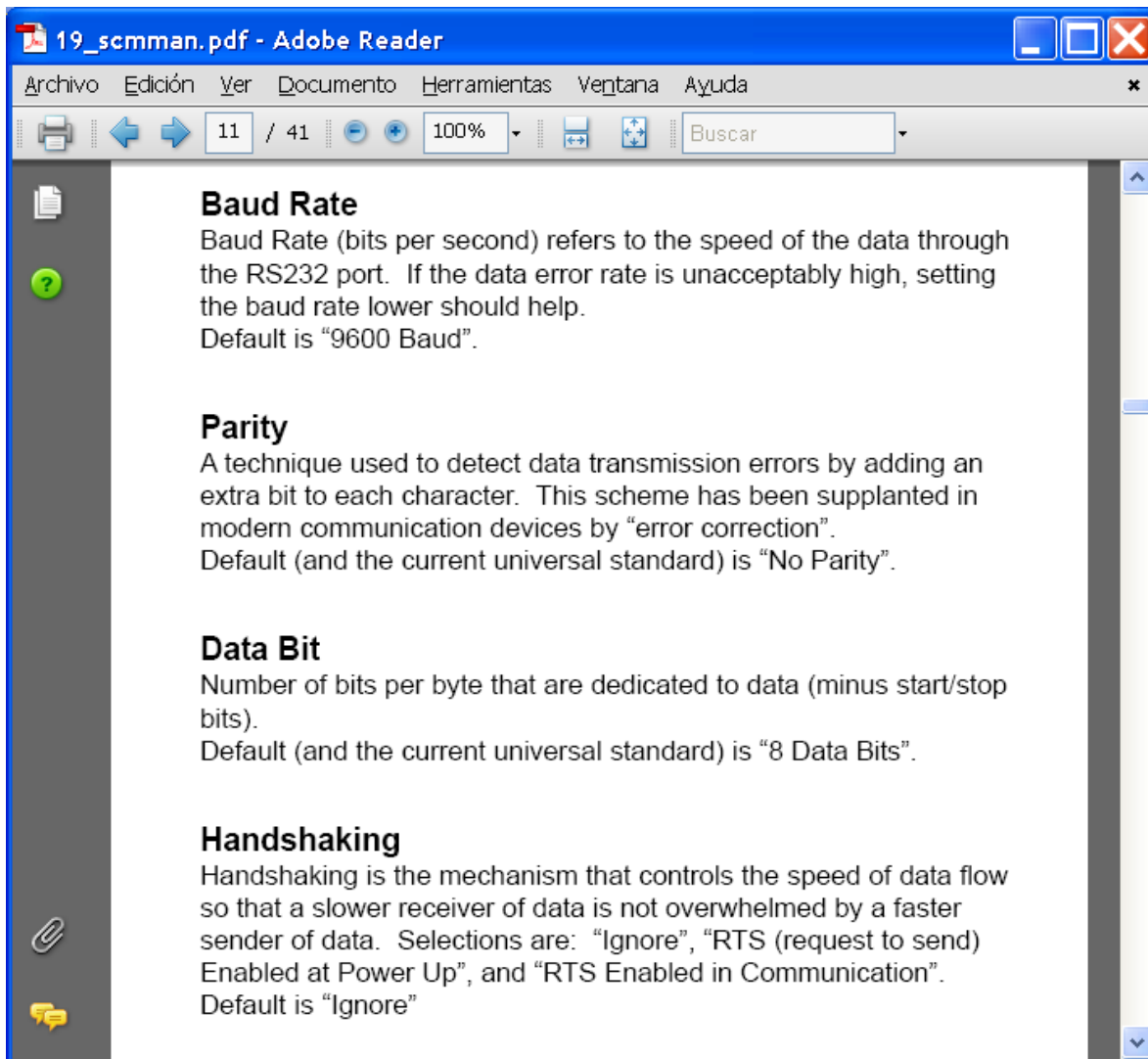


Figura 5.12 Configuración predeterminada del lector de código de barras. Tomado del documento 19\_scmman.pdf publicado por la empresa Unitech®

Por último se agregó un manejador de eventos para el evento *DataReceived* del objeto *puerto*. Este evento se disparará automáticamente cuando el lector de código de barras envíe información a la Terminal, esto es, cuando un empleado deslice su credencial por la ranura del lector.

Dentro del manejador del evento *DataReceived* se obtiene la cadena de caracteres representada por el código de barras que está impreso en la credencial. A continuación se muestra el código dentro del manejador del evento:

```
If puerto.BytesToRead > 0 T  
' Espera un poco para que teminen de llegar todos los bytes
```



```

Delay(100)
' Lee lo que hay en el puert
num = puerto.BytesToRead
cad = puerto.ReadLine
cad = Left(cad, num - 2)
num -= 2
'Codigo de barras recibido mediante el lector
nCodBarras = cad
If objLectora.estatus = EstatusLectora.AdminHD Then
    objLectora.estadoAdmin = EstatusAdmin.ConfirmandoNumer
End If
' Dispara el evento
RaiseEvent OnLectorCodigoBarras(num, cad)
End If

```

Cuando se ejecuta este evento, primero se realiza una pequeña espera de 0.2 segundos para dar tiempo a los dispositivos de terminar de transmitir toda la información. Esto con el fin de leer toda la cadena de caracteres representados por el código de barras, sin que falte ni un solo carácter. El tiempo es muy pequeño pues solo se trata de unos cuantos bytes.

Después se obtiene la cadena de caracteres que ya se encuentra dentro del buffer de lectura del puerto serie, y se le quita el fin de línea. Esta cadena de caracteres, que en el caso de ésta empresa consiste en un conjunto de 9 o 10 dígitos, son asignados a la variable global *nCodBarras*. De ésta forma, una vez habiendo terminado de ejecutar la rutina, el número del empleado impreso en el código de barras de la credencial estará disponible por los objetos encargados de procesarlo.

Finalmente, se actualiza el estatus de una de las pantallas de la aplicación, siempre y cuando la pantalla esté activa en ese momento, y se dispara un evento propio de la clase *LectorCodigoBarras*. La finalidad de disparar dicho evento es que el objeto o pantalla de la aplicación que esté utilizando el lector de código de barras en ese momento se percate de que se ha deslizado una credencial en la ranura del lector, procediendo entonces a procesar el número de empleado recibido.

### 5.6.7 La Interfaz para Manipular los Lectores de Huella Digital: *ILectorHuellaDigital*

Esta interfaz tiene como propósito especificar el conjunto de métodos que cualquier clase relacionada con algún módulo de huella digital en particular debe cumplir. Esta interfaz se ideó porque los módulos lectores de huellas digitales de diferentes fabricantes tienen conjuntos de instrucciones diferentes,

pero viendo los módulos desde una perspectiva más abstracta todos deben hacer lo mismo, sin importar como funcionen internamente.

A continuación se muestra la codificación de la interfaz encargada de registrar sobre el conjunto de operaciones abstractas que cualquier clase ligada a un módulo de huella digital debe ser capaz de realizar:

```
Public Interface IlectorHuellaDigital

    Function Activar() As Estatus
    Sub Desactivar()

    Function Enroll(ByVal id As Long) As Estatus

    Function DeleteAll() As Estatus
    Function DeleteID(ByVal id As Long) As Estatus
    Function DeleteIDIndex(ByVal id As Long, ByVal index As Short)
                                                As Estatus

    Function Verify(ByVal id As Long) As Estatus
    Function Identify(ByRef id As Long, ByRef index As Short) As Estatus

    Function IsFinger() As Boolean

    Function Copiar_Modulo_Lectora(ByVal Id As Long,
                                    ByVal Index As Short) As Boolean
    Function Copiar_Lectora_Modulo(ByVal Id As Long,
                                    ByVal Index As Short) As Boolean

    Sub Reset()
    Function Calibrar() As Estatus

End Interface
```

Los métodos *Activar* y *Desactivar* se utilizarán para encender y apagar el módulo de huella digital, respectivamente. El método *Enroll* es con el cual se grabarán nuevas huellas digitales dentro de los módulos. Los métodos *DeleteAll*, *DeleteId* y *DeleteIndex* se encargarán de eliminar huellas digitales de aquellas personas que ya no laboren en la empresa.

Los métodos *Verify* e *Identify* serán los encargados de autenticar a los empleados que registren su entrada o salida. *IsFinger* se utilizará para saber si alguna persona colocó su dedo en el sensor del lector de huellas digitales.

Los dos métodos cuyos nombres tienen el prefijo “*Copiar*”, serán los encargados de copiar archivos de huella digital de la Terminal al módulo y viceversa. Los métodos *Reset* y *Calibrar* se utilizarán, como su nombre lo

indica, para realizar un reestablecimiento del módulo y para calibrar el sensor que obtiene la imagen de la huella digital. Su finalidad es reestablecer y ajustar el módulo cuando por algún motivo éste genera algún error y se vuelve inestable.

Una vez definido el conjunto de rutinas mínimo que cualquier módulo de huella digital debe cumplir, ya se puede proceder a su codificación. Esto permite, en un momento dado, volver a utilizar de nuevo el módulo de huella digital que viene de fábrica en las terminales, o cualquier otro módulo de huella digital que de alguna forma se pueda colocar dentro de la carcasa de la Terminal, o incluso aunque se tuviera que colocar por fuera y conectarse a la misma, por ejemplo, vía USB. Lo único que se necesitaría en lo que se refiere al software, sería una clase específicamente diseñada para trabajar con ese módulo. El resto de la aplicación no sufriría ningún cambio, pues en cualquier parte donde se utiliza el lector de huellas digitales se hace a través de la interfaz, no de la clase que lo controla.

Esto no significa que todos los módulos de huella digital que se puedan utilizar con la Terminal deban poseer la capacidad de realizar todas estas funcionalidades, sino que habrá algunas funciones que un cierto módulo no posea y en ese caso la rutina dentro de su clase se implementará de forma vacía.

#### **5.6.8 La Clase que Manipula el Módulo SFM3500: HD\_SFM3500**

La funcionalidad de este módulo se programó utilizando comunicación serial. A continuación se explica su funcionamiento.

La primera rutina de esta clase es el constructor, donde se crea un objeto de tipo *SerialPort* que se utilizará precisamente para comunicarse con el módulo en todas y cada una de las instrucciones. Enseguida se muestra su código:

```
Sub New()  
  
    Delay(1000)  
  
    ' Create a new SerialPort object with default settings.  
    puerto = New SerialPort()  
  
    puerto.PortName = "COM6"  
    puerto.BaudRate = 115200  
    puerto.Parity = Parity.None  
    puerto.DataBits = 8
```

```

    puerto.StopBits = StopBits.One
    puerto.Handshake = Handshake.None

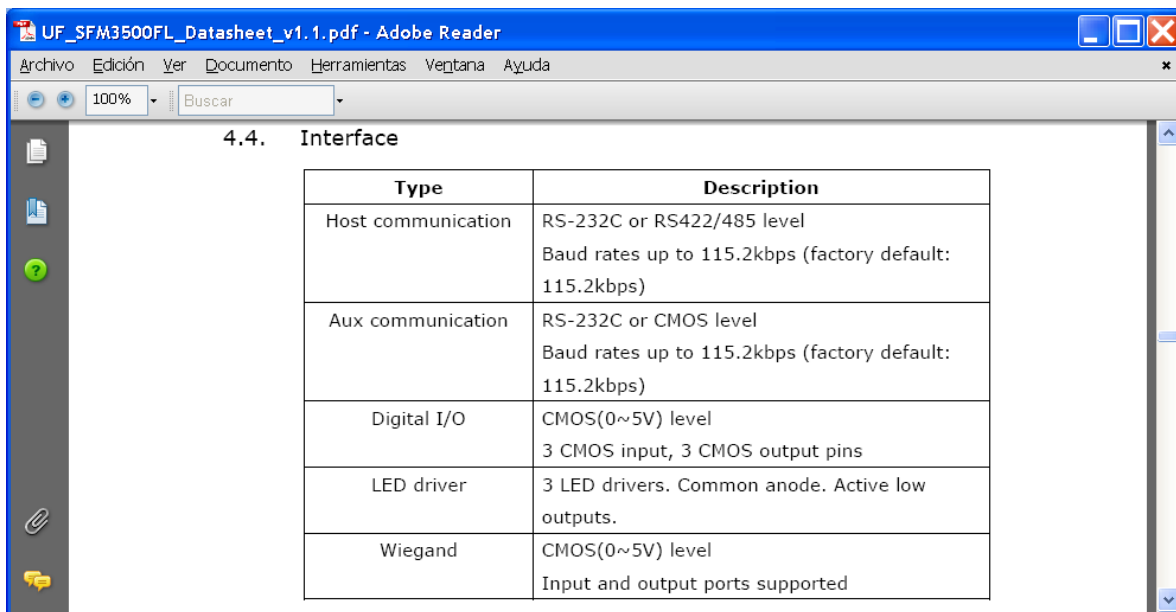
    ' Set the read/write sizes
    puerto.ReadBufferSize = 1000
    puerto.WriteBufferSize = 1000

    ' Manejador del puerto
    AddHandler puerto.DataReceived, AddressOf DataReceived

End Sub

```

Los parámetros del puerto serie también se tomaron de la hoja de datos del módulo. En la figura 5.13 se muestra un fragmento de la configuración que traen de fábrica esos módulos.



4.4. Interface

Type	Description
Host communication	RS-232C or RS422/485 level Baud rates up to 115.2kbps (factory default: 115.2kbps)
Aux communication	RS-232C or CMOS level Baud rates up to 115.2kbps (factory default: 115.2kbps)
Digital I/O	CMOS(0~5V) level 3 CMOS input, 3 CMOS output pins
LED driver	3 LED drivers. Common anode. Active low outputs.
Wiegand	CMOS(0~5V) level Input and output ports supported

Figura 5.13 Configuración de fábrica del módulo SFM3500. Tomado del documento UF\_SFM3500FL\_Datasheet\_v1.1.pdf publicado por la empresa Suprema®

También dentro del constructor de la clase se agrega un manejador para el evento de recepción de datos del puerto serie. Antes de explicar el resto de las rutinas se explicará el protocolo que utiliza el módulo para comunicarse con la Terminal.

Para enviar y recibir información el módulo SFM3500 se basa en paquetes de bytes. Estos paquetes tienen la estructura que se muestra en la figura 5.14.

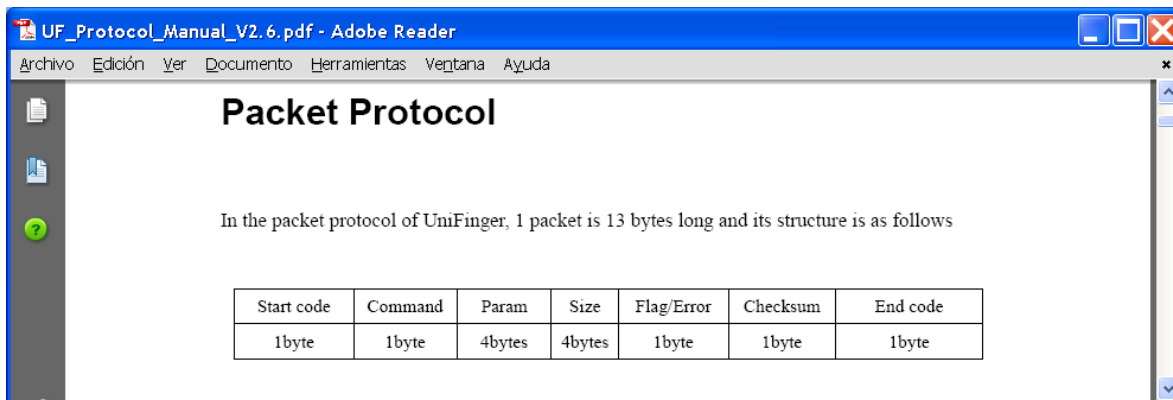


Figura 5.14 Estructura del paquete de datos del módulo SFM3500. Tomado del documento *UF\_Protocol\_Manual\_V2.6.pdf* publicado por la empresa Suprema®

Todas las instrucciones utilizan esta estructura para enviar y recibir información con el módulo. A continuación se describen cada uno de estos campos:

1. **Start Code.** Indica el inicio del paquete. Tiene un valor de 0x40.
2. **Command.** Es el código del comando que se va a ejecutar.
3. **Param.** Es un campo de parámetro que se utiliza en la mayoría de las instrucciones.
4. **Size.** Indica el tamaño de la información que se transmitirá, tal como archivos de huella digital o imágenes. En muchas instrucciones se utiliza también como campo de parámetro.
5. **Flag/Error.** En la solicitud se utiliza como campo de parámetro y en la respuesta indica el resultado de la operación (éxito o error).
6. **Checksum.** Es un campo de validación de errores. Es el residuo de la sumatoria de todos los campos desde **Start Code** hasta **Flag/Error**, dividido entre 0x100.
7. **End Code.** Indica el fin del paquete. Tiene un valor de 0x0A.

Para manipular más fácilmente esta estructura se ha creado una parecida en el programa:

```
Private Structure Packet
    Dim Command As Byte
    Dim Param0 As Byte
    Dim Param1 As Byte
```

```

Dim Param2 As Byte
Dim Param3 As Byte
Dim Size0 As Byte
Dim Size1 As Byte
Dim Size2 As Byte
Dim Size3 As Byte
Dim Flag_Error As Byte
End Structure

```

Solamente faltan aquí los campos *Start Code*, *End Code* y *Checksum*, por ser fijos los primeros dos y calculado en tiempo real el tercero. En el caso de *Start Code* y *End Code*, se han declarado dos constantes para poder identificarlas más fácilmente durante el desarrollo de la aplicación:

```

Private Const START_CODE As Integer = &H40
Private Const END_CODE As Integer = &HA

```

En el caso de los campos *Param* y *Size* se han definido dos rutinas para asignarles sus valores correspondientes, así como otras dos rutinas para obtener el valor que almacenan, pues se trata de números que deben almacenarse en cuatro bytes. A continuación se muestran las rutinas encargadas de asignarles valores a esos campos.

```

Private Sub setParam(ByVal valor As Long)

    pac.Param0 = valor And &HFF
    pac.Param1 = (valor And &HFF00) / &H100
    pac.Param2 = (valor And &HFF0000) / &H10000
    pac.Param3 = (valor And &HFF000000) / &H1000000

End Sub

Private Sub setSize(ByVal valor As Long)

    pac.Size0 = valor And &HFF
    pac.Size1 = (valor And &HFF00) / &H100
    pac.Size2 = (valor And &HFF0000) / &H10000
    pac.Size3 = (valor And &HFF000000) / &H1000000

End Sub

```

En ambos casos, se debe dividir el número recibido en cuatro bytes, y colocar cada byte resultante en su posición correspondiente. El producto “booleano” con los múltiplos de 0xFF permite aislar cada uno de los cuatro bytes, mientras que la división entre los múltiplos de 0x01 permite acomodar el valor de cada byte en la posición adecuada.

A continuación, se muestran las codificaciones de las rutinas que obtienen los valores de esos campos.

```
Private Function getParam() As Long

    Dim valor As Long = 0

    valor = pac.Param0
    valor += pac.Param1 * &H100
    valor += pac.Param2 * &H10000
    valor += pac.Param3 * &H1000000

    Return valor

End Function

Private Function getSize() As Long

    Dim valor As Long = 0
    valor = pac.Size0
    valor += pac.Size1 * &H100
    valor += pac.Size2 * &H10000
    valor += pac.Size3 * &H1000000

    Return valor

End Function
```

Se trata del proceso contrario: a partir de la unión de cuatro bytes se obtiene el número que representan. Para lograrlo se multiplica cada byte por un múltiplo de 0x01, con lo cual se obtiene la cantidad real que representa ese byte, y se suman los resultados obtenidos de cada multiplicación.

El motivo por el cual se tiene que hacer todo esto en vez de almacenar el número directamente como entero largo (*Long*) es porque al momento de realizar el envío del paquete, el orden de los bytes transmitidos es de tipo “*little endian*”, en el cual primero se transmite el byte de menor orden significativo. Si se almacenara el número de cuatro bytes tal cual, al momento de transmitirlo no se cumpliría ésta condición, produciéndose envíos de información errónea.

En la figura 5.15 se muestra un fragmento de la documentación que trata acerca de la transmisión de los bytes:

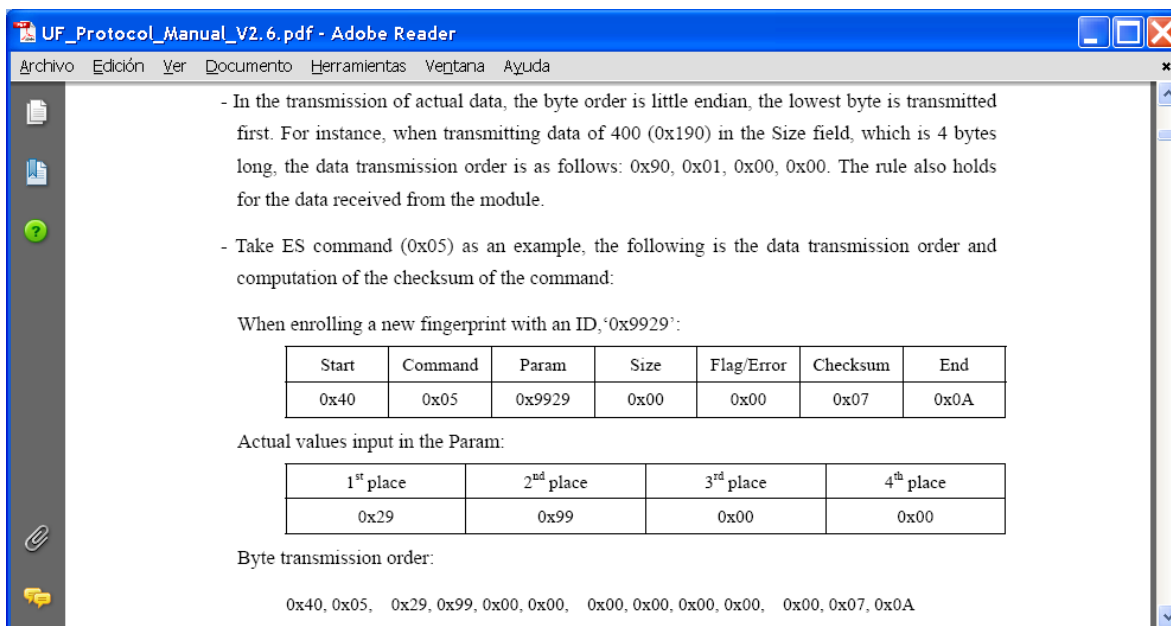


Figura 5.15 Orden de transmisión de los bytes. Tomado del documento *UF\_Protocol\_Manual\_V2.6.pdf* publicado por la empresa Suprema®

En resumen lo que se explica en este párrafo de la documentación del módulo es que si se debe transmitir, por ejemplo, el número 190Hex, este se almacena internamente en 4 bytes de datos:

00Hex, 00Hex, 01Hex, 90Hex

Al momento de ser transmitidos debe hacerse en el siguiente orden:

90Hex, 01Hex, 00Hex, 00Hex

De igual forma, si se debe transmitir el número 9929Hex, este se encuentra almacenado como:

00Hex, 00Hex, 99Hex, 29Hex

El orden correcto de transmisión deberá ser:

29Hex, 99Hex, 00Hex, 00Hex

Para calcular el valor del campo *Checksum* primero se tienen que sumar los valores de cada byte del paquete desde el campo *Start Code* hasta el campo



*Flag/Error*, y posteriormente se toma el residuo de dividir el resultado de la sumatoria entre 0x100.

En la figura 5.16 se muestra la documentación correspondiente:

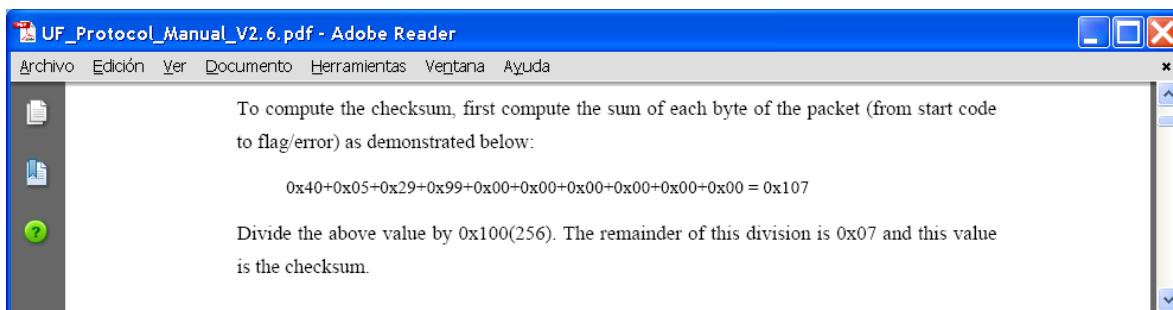


Figura 5.16 Cálculo del campo CheckSum. Tomado del documento *UF\_Protocol\_Manual\_V2.6.pdf* publicado por la empresa Suprema®

La rutina que se codificó para realizar éste cálculo se muestra a continuación:

```
Private Function calcChecksum() As Long

    Dim tot As Long = 0
    tot = START_CODE
    tot += pac.Command
    tot += pac.Flag_Error
    tot += pac.Param0
    tot += pac.Param1
    tot += pac.Param2
    tot += pac.Param3
    tot += pac.Size0
    tot += pac.Size1
    tot += pac.Size2
    tot += pac.Size3

    Return tot Mod &H100

End Function
```

Como puede observarse, de cualquier forma resulta necesario separar los cuatro bytes de los campos *Param* y *Size*, pues la sumatoria para calcular el *Checksum* se realiza a nivel de bytes.

Por último se necesitan dos rutinas más: una para obtener toda la secuencia de bytes del paquete de forma continua lista para transmitirse por el puerto serie, y otra para armar una estructura completa a partir de los bytes obtenidos como respuesta en el mismo puerto.

La primera de ellas se muestra a continuación:

```
Private Function getPacketBytes() As Byte()  
  
    Dim x(_TAMAÑO_PACKET - 1) As Byte  
  
    x(0) = START_CODE  
  
    x(1) = pac.Command  
    x(2) = pac.Param0  
    x(3) = pac.Param1  
    x(4) = pac.Param2  
    x(5) = pac.Param3  
    x(6) = pac.Size0  
    x(7) = pac.Size1  
    x(8) = pac.Size2  
    x(9) = pac.Size3  
    x(10) = pac.Flag_Error  
  
    x(11) = Me.calcChecksum()  
  
    x(12) = END_CODE  
  
    Return x  
  
End Function
```

En esta rutina primero se declara un arreglo de bytes del tamaño del paquete y después se asignan cada uno de los campos de la estructura en las posiciones adecuadas dentro del arreglo.

La rutina que armará un paquete a partir de los datos recibidos por el puerto serie se muestra a continuación:

```
Private Sub setPacketBytes(ByVal x As Byte())  
  
    pac.Command = x(1)  
    pac.Param0 = x(2)  
    pac.Param1 = x(3)  
    pac.Param2 = x(4)  
    pac.Param3 = x(5)  
    pac.Size0 = x(6)  
    pac.Size1 = x(7)  
    pac.Size2 = x(8)  
    pac.Size3 = x(9)  
    pac.Flag_Error = x(10)  
  
End Sub
```

Aquí solamente se toma el valor de cada byte recibido y se coloca en el campo correspondiente de la estructura.

Para algunas rutinas de la clase se han utilizado dos constantes que representan el tamaño del paquete y el tamaño de una huella digital. Ambas constantes se muestran a continuación:

```
Private Const _TAMAÑO_TEMPLATE As Integer = 384
Private Const _TAMAÑO_PACKET As Integer = 13
```

Una vez preparadas las rutinas que manipularán correctamente los paquetes de información, se puede proseguir a codificar cada una de las funciones que se necesitarán del módulo de huella digital, basándose para ello en la hoja de datos del mismo. En general casi todas las funciones tienen la misma estructura y de hecho son casi idénticas, difiriendo únicamente en los parámetros que reciben. A continuación se explican dos ejemplos de la codificación de estas funciones.

La primera función que se explicará es la encargada de eliminar una huella digital del módulo. Su codificación se muestra enseguida:

```
Private Sub deleteTemplate(ByVal id As Integer, ByVal index As Byte)

    _estatus = EstatusLector.Ocupado

    ' Borra una huella
    comando = DELETE_TEMPLATE
    pac.Command = DELETE_TEMPLATE
    Me.setParam(id)
    Me.setSize(index)
    pac.Flag_Error = _DELETE_ONLY_ONE

    ' Lo manda
    puerto.Write(Me.getPacketBytes(), 0, _TAMAÑO_PACKET)

End Sub
```

En primer lugar, se utiliza una variable miembro de la clase para controlar el acceso al módulo, de forma que se accederá a él para darle una sola instrucción a la vez. Posteriormente se asignan los parámetros correspondientes a la instrucción en una estructura de tipo *Packet*, la cual finalmente es enviada por el puerto serie utilizando el método *Write* del objeto *puerto*. Es al momento de realizar el envío cuando se obtienen los bytes del paquete de datos que se construyó con las instrucciones anteriores.

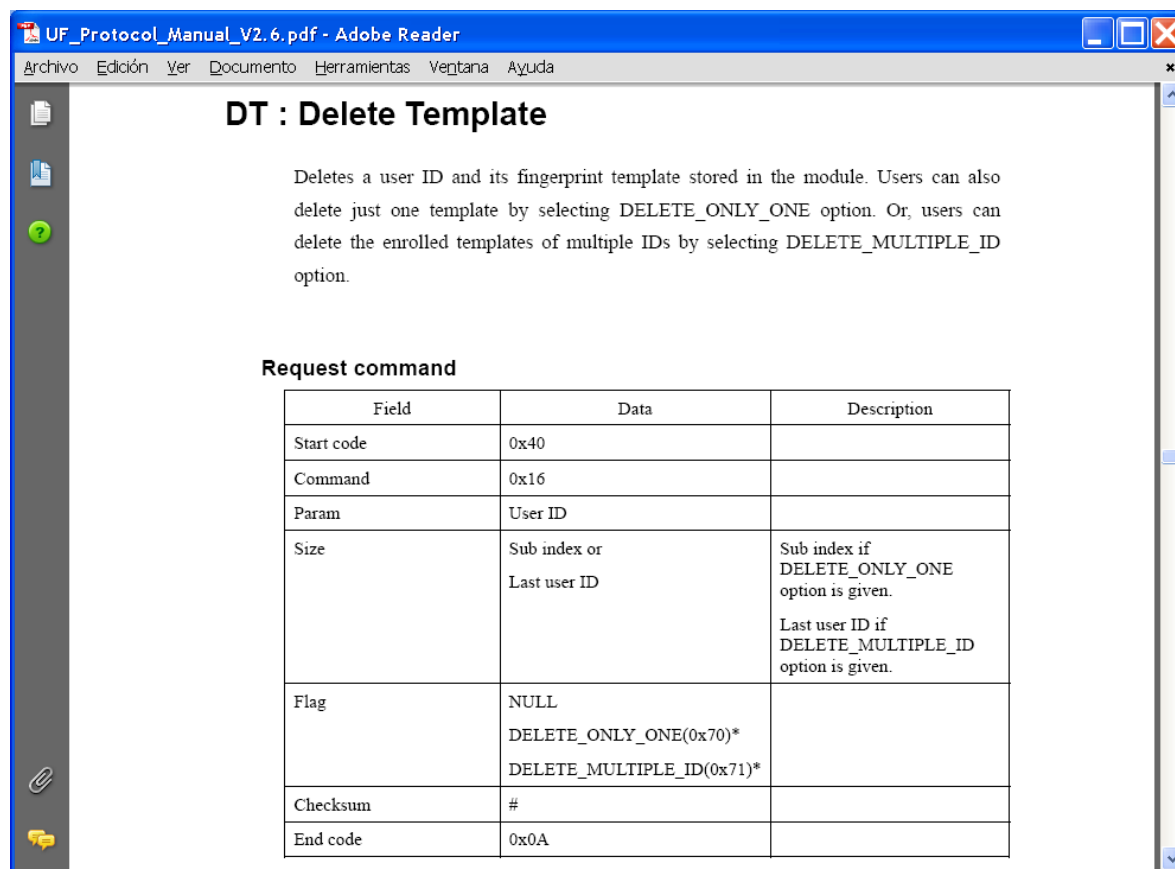
Los parámetros que se deben enviar dentro del paquete se obtienen de la hoja de datos del módulo. En las figuras 5.17 y 5.18 se muestra la documentación de la función *deleteTemplate*.

Como se puede observar, se debe asignar al paquete los códigos de inicio y de fin (*Start Code* y *End code*), los cuales se asignan con las rutinas explicadas en párrafos anteriores. También se debe especificar el número de comando. Para este fin, dentro de la clase se ha declarado una constante para cada instrucción que se requiera utilizar. En este caso se trata de la siguiente declaración:

```
Private Const DELETE_TEMPLATE As Integer = &H16
```

Después se utilizan los campos *Param*, *Size* y *Flag* para enviar los parámetros adicionales de la función. Para esta instrucción los parámetros son: El número del empleado, el número de huella de ese empleado, y un parámetro extra que le indica al módulo como debe realizar la eliminación. De igual forma se han declarado constantes para una mejor legibilidad:

```
Private Const _DELETE_ONLY_ONE As Integer = &H70
```



**DT : Delete Template**

Deletes a user ID and its fingerprint template stored in the module. Users can also delete just one template by selecting `DELETE_ONLY_ONE` option. Or, users can delete the enrolled templates of multiple IDs by selecting `DELETE_MULTIPLE_ID` option.

**Request command**

Field	Data	Description
Start code	0x40	
Command	0x16	
Param	User ID	
Size	Sub index or Last user ID	Sub index if <code>DELETE_ONLY_ONE</code> option is given.  Last user ID if <code>DELETE_MULTIPLE_ID</code> option is given.
Flag	NULL <code>DELETE_ONLY_ONE(0x70)*</code> <code>DELETE_MULTIPLE_ID(0x71)*</code>	
Checksum	#	
End code	0x0A	

Figura 5.17 Documentación de la instrucción *Delete Template*. Tomado del documento *UF\_Protocol\_Manual\_V2.6.pdf* publicado por la empresa Suprema®

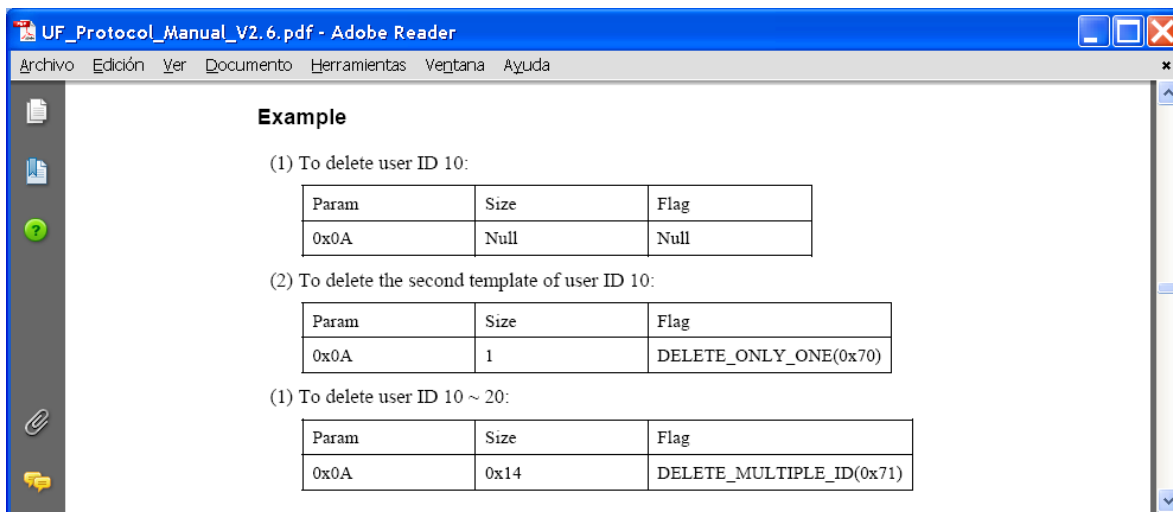


Figura 5.18 Documentación de la instrucción *Delete Template* (continuación). Tomado del documento *UF\_Protocol\_Manual\_V2.6.pdf* publicado por la empresa Suprema®

El valor del campo *Checksum* se calcula utilizando la función *calcChecksum*. Otro ejemplo de una función es la encargada de verificar si una huella digital corresponde con el número de empleado proporcionado. Su codificación se muestra enseguida:

```
Private Sub identifyByScan()

    _estatus = EstatusLector.Ocupado

    ' Verifica una huella
    comando = IDENTIFY_BY_SCAN
    pac.Command = IDENTIFY_BY_SCAN
    Me.setParam(NULL)
    Me.setSize(NULL)
    pac.Flag_Error = NULL

    ' Lo manda
    puerto.Write(Me.getPacketBytes(), 0, _TAMAÑO_PACKET)

End Sub
```

Esta rutina es casi idéntica a la rutina encargada de eliminar las huellas (*deleteTemplate*) que se describió anteriormente. La única diferencia son los parámetros que recibe cada una. En la figura 5.19 se muestra la documentación correspondiente:

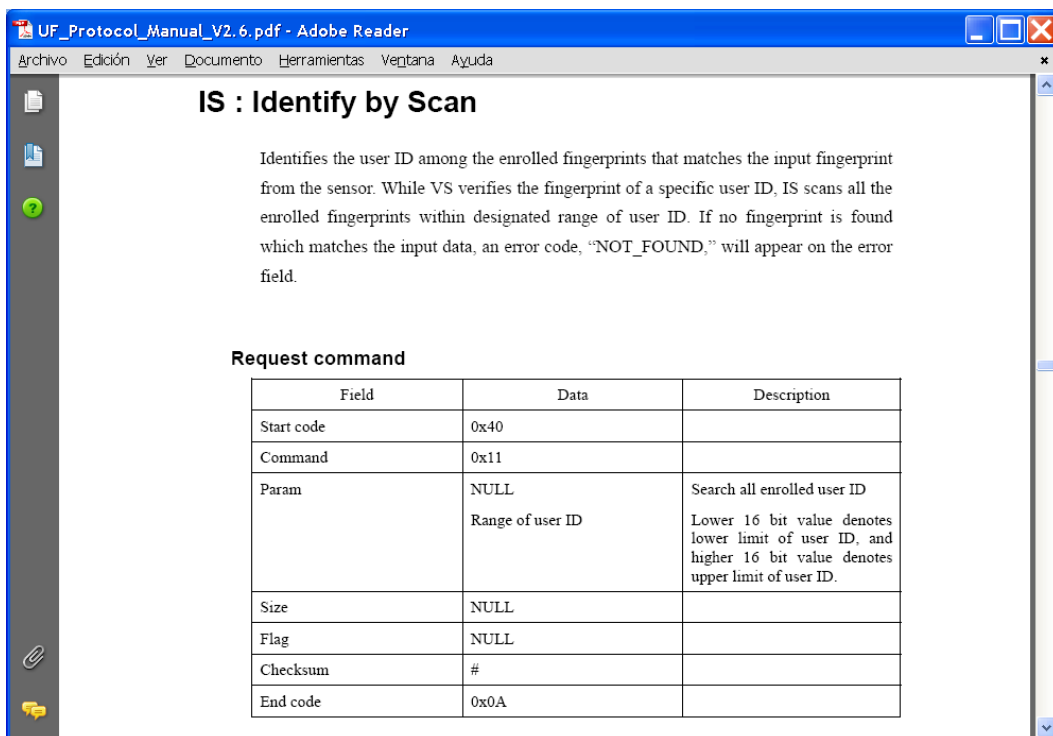


Figura 5.19 Documentación de la instrucción Identify by Scan. Tomado del documento UF\_Protocol\_Manual\_V2.6.pdf publicado por la empresa Suprema®

Además de las funcionalidades para manipular las huellas digitales que ofrece el módulo, también se puede ajustar mediante algunos parámetros. Por ejemplo, se le puede indicar si la Terminal estará dentro de alguna instalación o si estará expuesta a la luz solar, de forma que el módulo trabaje con el sensor de huellas de la forma más adecuada. O se le puede indicar el tiempo que el módulo deberá esperar entre dos reconocimientos de huellas continuos, de forma que si un empleado coloca su dedo en el sensor y se tarda un poco en quitarlo, el módulo no realice un doble o un triple reconocimiento.

Estos parámetros se ajustan mediante una función del módulo que también es casi idéntica a las ya descritas. A continuación se muestra su codificación:

```
Public Sub setSystemParameter(ByVal param As Integer,
                               ByVal valor As Integer)
    _estatus = EstatusLector.Ocupado

    ' Asigna un parámetro del módulo
    comando = SYSTEM_PARAMETER_WRITE
    pac.Command = SYSTEM_PARAMETER_WRITE
    Me.setParam(NULL)
    Me.setSize(valor)
    pac.Flag_Error = param
End Sub
```

```

' Lo manda
puerto.Write(Me.getPacketBytes(), 0, _TAMAÑO_PACKET)

Me.EsperarRespuesta()

End Sub

```

Lo único que tiene de especial esta rutina es que uno de los parámetros corresponde con un parámetro ajustable del módulo. De igual forma, se han declarado constantes para facilitar su uso:

```

Private Const SYSTEM_PARAMETER_WRITE As Integer = &H1

Public Const _LIGHTING_CONDITION As Integer = &H90

Public Const _FREE_SCAN_DELAY As Integer = &H91

```

En general, todas las rutinas relacionadas con la manipulación del módulo de huella digital son casi idénticas y se utilizan exactamente de la misma forma. La única excepción es la rutina encargada de grabar dentro del módulo una huella digital que se ha obtenido desde el servidor vía el Servicio Web, y su pseudo código se muestra a continuación:

```

Private Sub enrollByTemplate(ByVal id As Integer, ByVal index As Byte)

    <Asignar el estatus del lector como "Ocupado">

    <Armar una estructura de tipo "Packet" con los bytes necesarios
    para realizar la grabación de la huella dentro del módulo>

    <Iniciar bloque Try-Catch de control de errores>

        <Leer los datos del archivo de la huella digital y colocarlos
        en un arreglo de bytes>

    <Terminar bloque Try-Catch de control de errores>

    <Enviar por el puerto serie la estructura de tipo "Packet" y el
    arreglo de bytes>

    <Devuelve Éxito ó Error como resultado de la función>

End Sub

```

**Ver Anexo B.4**

La diferencia aquí es que además de asignar los valores correspondientes dentro del paquete, también se debe leer del disco el archivo que contiene la huella digital. Estos datos son almacenados en forma de un arreglo de bytes y

enviados inmediatamente después del paquete. La documentación se muestra en las figuras 5.20, 5.21 y 5.22.

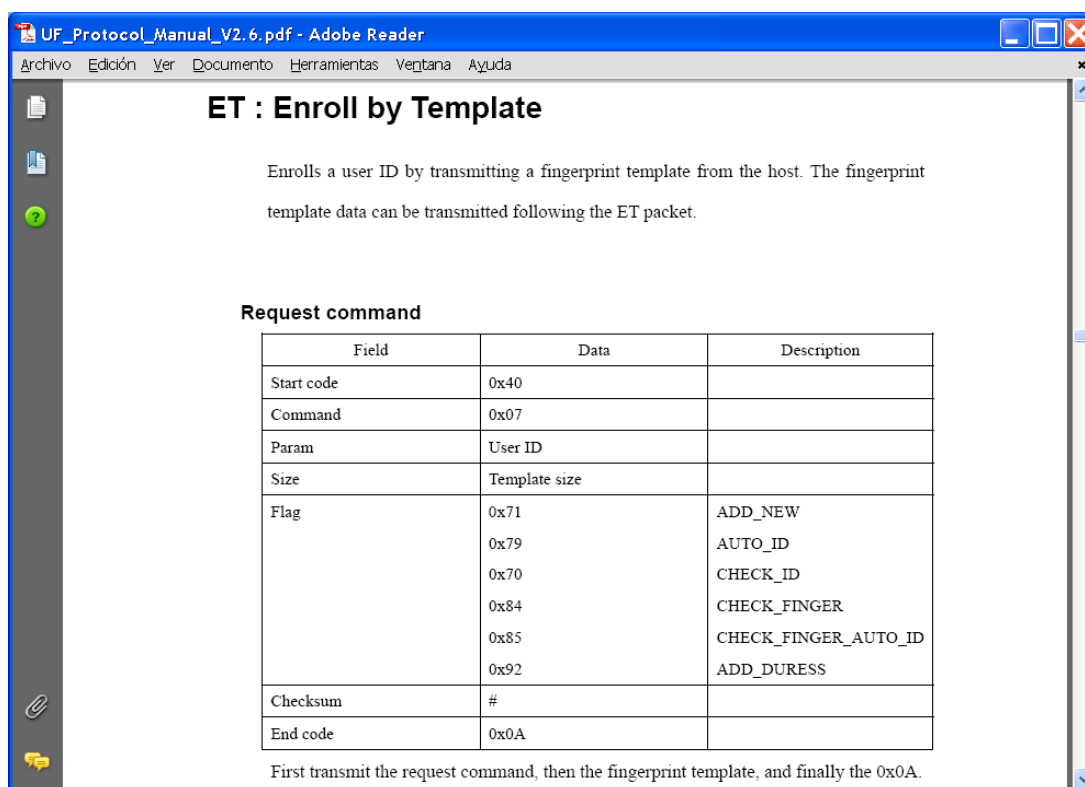


Figura 5.20 Documentación de la instrucción Enroll by Template. Tomado del documento UF\_Protocol\_Manual\_V2.6.pdf publicado por la empresa Suprema®

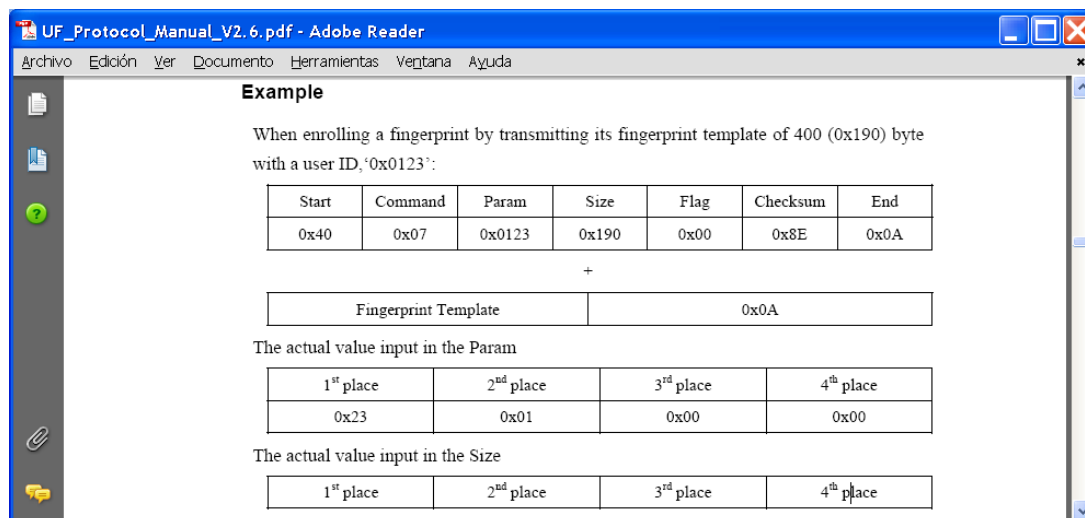


Figura 5.21 Documentación de la instrucción Enroll by Template (continuación). Tomado del documento UF\_Protocol\_Manual\_V2.6.pdf publicado por la empresa Suprema®



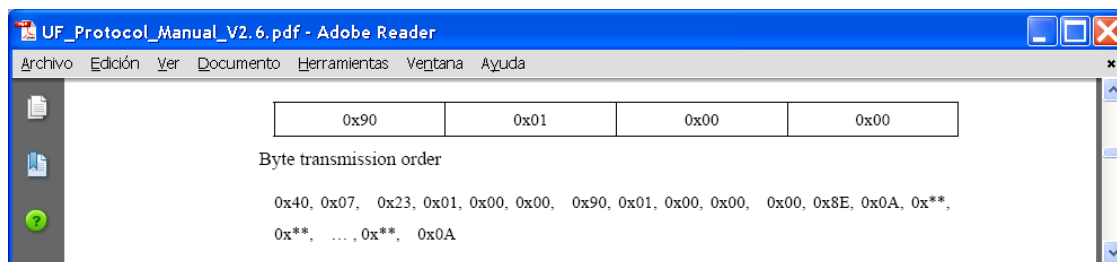


Figura 5.22 Documentación de la instrucción Enroll by Template (continuación). Tomado del documento UF\_Protocol\_Manual\_V2.6.pdf publicado por la empresa Suprema®

Todo este conjunto de rutinas que manipulan directamente el módulo de huella digital se podrían utilizar desde cualquier parte de la aplicación. Sin embargo, conviene crear un segundo conjunto de rutinas que sirven como envoltorios de las primeras, esto con el fin de colocar ahí todo el código extra que se necesite para controlar el estado de la aplicación durante el tiempo que el módulo se tarde procesando cada instrucción que se le ordene.

Es por este motivo que las instrucciones descritas anteriormente comienzan con la instrucción:

```
_estatus = EstatusLector.Ocupado
```

Se trata de una variable miembro que puede tener cualquiera de dos valores definidos con una enumeración:

```
Private Enum EstatusLector
    Normal
    Ocupado
End Enum
```

El objetivo es controlar la aplicación para que no se intente ordenar una instrucción al módulo cuando éste se encuentre ya procesando alguna. Por lo tanto, esta variable de control cambiará su valor cuando el módulo termine de procesar la instrucción indicada y devuelva una respuesta a la Terminal.

Teniendo esto en mente, se puede codificar el conjunto de rutinas que utilizará a las primeras. Así, por ejemplo, para eliminar las huellas de un empleado se ha creado la siguiente rutina:

```
Public Function DeleteID(ByVal id As Long) As Estatus
    Implements IlectorHuellaDigital.DeleteID

    Dim res1, res2 As Estatus
```

```

' Ejecuta y espera la respuesta
Me.deleteTemplate(id, 0)
EsperarRespuesta()
res1 = resp

' Ejecuta y espera la respuesta
Me.deleteTemplate(id, 1)
EsperarRespuesta()
res2 = resp

If res1 = [ERROR] And res2 = [ERROR] Then
    Return [ERROR]
End If

Return EXITO

End Function

```

En esta rutina se utiliza el método *deleteTemplate* que se explicó anteriormente para eliminar las huellas 0 y 1 del empleado. En esta aplicación basta con hacerlo así porque solamente se grabará un máximo de dos huellas por persona. Después se ejecuta una rutina llamada *EsperarRespuesta*, la cual se encargará de detener la ejecución del programa mientras el módulo termina de procesar la instrucción solicitada.

Finalmente, en la variable miembro *resp* se obtiene el resultado de la operación: “Éxito” o “Error”, así la rutina devolverá a su vez un estatus de “Éxito” o “Error” dependiendo de la combinación de ambos resultados.

Otro ejemplo de este conjunto de rutinas es el que realiza una identificación de una huella digital. En este caso, se solicita al módulo que realice la identificación y devuelva el número del empleado del que se trata. El código de esta rutina se muestra a continuación:

```

Public Function Identify(ByRef id As Long, ByRef index As Short)
    As Estatus Implements IlectorHuellaDigital.Identify
    ' Ejecuta y espera la respuesta
    Me.identifyByScan()

    EsperarRespuesta()

    id = Me.id
    index = Me.index

    Return resp
End Function

```

Aquí, de igual forma, se ejecuta la rutina correspondiente y se invoca la función encargada de esperar hasta que el módulo termine. Cuando el módulo devuelve un resultado, éste es almacenado en los parámetros recibidos por referencia, para que puedan ser utilizados desde fuera de la clase.

En general, todo este conjunto de rutinas es similar en el sentido de que invocan a la rutina correspondiente para manipular directamente al módulo e invocan también la rutina encargada de esperarlo, para devolver finalmente el resultado de la operación.

A continuación se muestra el código de la rutina de espera:

```
' rutina para esperar respuesta
Private Sub EsperarRespuesta()
    Do
        Application.DoEvents()
    Loop While _estatus = EstatusLector.Ocupado
End Sub
```

Como puede observarse, se trata de un simple bucle que se ejecuta indefinidamente mientras el estatus del módulo sea “ocupado”. Sin embargo, incluye una instrucción que permite la ejecución de eventos del sistema, y es precisamente mediante un evento que se recibe la respuesta del módulo y se asigna de nuevo el valor de “normal” a la variable de control, permitiendo así que termine la ejecución del bucle y el programa continúe donde se quedó.

Cuando el módulo termina de procesar la instrucción solicitada por la Terminal, envía su resultado también por el puerto serie, el cual es recibido por la Terminal utilizando el código que se muestra enseguida:

```
Private Sub DataReceived(ByVal sender As Object,
                        ByVal e As SerialDataReceivedEventArgs)

    If Me.procesandoRecepcion Then Exit Sub

    ' Para controlar la llegada continua y constante del evento
    Me.procesandoRecepcion = True
    Me.errorBytes = False

    Dim datos(_TAMAÑO_PACKET - 1) As Byte

    ' Espera los bytes
    Me.esperarAQueLleguenLosBytes(_TAMAÑO_PACKET)
    If errorBytes Then GoTo xFin

    ' Obtiene el paquete que llego
```

```
puerto.Read(datos, 0, _TAMAÑO_PACKET)
Me.setPacketBytes(datos)
```

```
End Sub
```

Esta es solo una parte del manejador del evento con el cual se reciben los datos enviados por el módulo. El resto de la rutina se explicará más adelante. En esta rutina se utiliza otra variable de control para que si por cualquier motivo el evento se dispara dos veces (probablemente debido a una recepción parcial de los datos seguida de una recepción de los datos complementarios), sólo se procese la primera hasta terminar de recibir todos los datos y se ignoren las demás.

Posteriormente, se apoya en una rutina auxiliar que consiste en una espera para que lleguen la cantidad de bytes que se deben recibir. Si en este proceso se detecta que no han llegado todos los bytes esperados, se continúa con la ejecución del programa donde más adelante se examinará el valor de la variable *errorBytes* y se tomarán las acciones necesarias para controlar este error.

Por el contrario, si los datos se han recibido correctamente, estos son asignados a sus respectivas posiciones dentro la estructura *Packet*, y se continúa el flujo normal del programa donde se obtendrá el valor devuelto por el módulo.

Esta es la rutina auxiliar encargada de esperar a que el módulo termine de enviar todos los bytes necesarios:

```
Private Sub esperarAQueLleguenLosBytes(ByVal cantidad As Long)

    ' Espera a que lleguen cierta cantidad de bytes
    intentosRecepcion = 0

    Do While puerto.BytesToRead < cantidad

        If puerto.BytesToRead < cantidad Then
            intentosRecepcion += 1
            If intentosRecepcion = 10 Then
                Me.errorBytes = True
                Exit Do
            End If
        End If

        Delay(20)

    Loop
```

```
intentosRecepcion = 0
```

```
End Sub
```

Aquí solamente se revisa el valor de la propiedad *BytesToRead* del objeto que representa al puerto serie, hasta que se han recibido los bytes necesarios. Se ha agregado un contador para evitar que el programa se quede indefinidamente ahí, de forma que si se llega a interrumpir la comunicación, el programa pueda continuar su ejecución y no se quede clavado ahí esperando una serie de bytes que nunca van a llegar.

Una vez que se han recibido los bytes necesarios y se han asignado a la estructura *Packet*, el manejador del evento de recepción de datos prosigue a determinar cual fue la instrucción que se ejecutó y obtener entonces la respuesta de forma correcta. A continuación se muestra un fragmento del código:

```
' Procesa la entrada para saber si hizo bien las cosas
Select Case pac.Command
    Case RESET_MODULE
        resp = Estatus.EXITO

    Case SYSTEM_PARAMETER_WRITE
        If pac.Flag_Error = SUCCESS Then
            resp = Estatus.EXITO
        Else
            resp = Estatus.ERROR
        End If

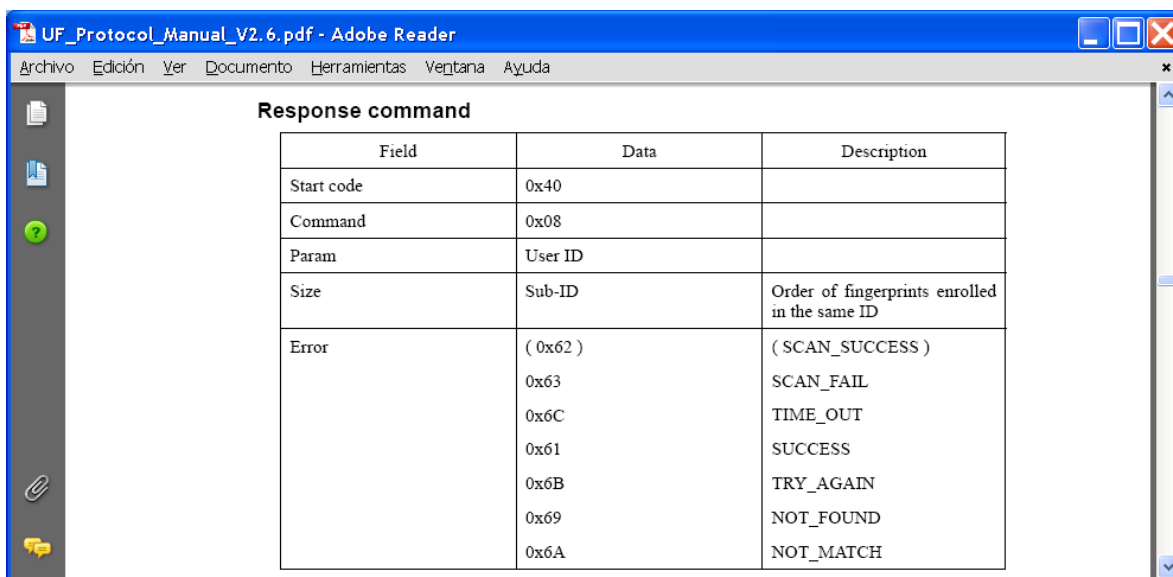
    Case SYSTEM_PARAMETER_SAVE
        If pac.Flag_Error = SUCCESS Then
            resp = Estatus.EXITO
        Else
            resp = Estatus.ERROR
        End If

    Case ENROLL_BY_SCAN
        If pac.Flag_Error = SUCCESS Then
            resp = Estatus.EXITO
            Me.quality = Me.getSize()
        Else
            resp = Estatus.ERROR
        End If
```

Se trata de una instrucción *Select Case* con la cual se examina el valor del campo *Command* de la estructura *Packet*. Una vez que se ha determinado la instrucción de la que se trata, se examina el campo *Flag/Error* para saber si se

trata de una respuesta exitosa o de un error. Finalmente, dependiendo de la instrucción, se obtiene el valor de uno u otro campo de la estructura. La documentación de la forma en que el módulo devuelve los valores dentro del paquete se puede ver en la hoja de datos del mismo.

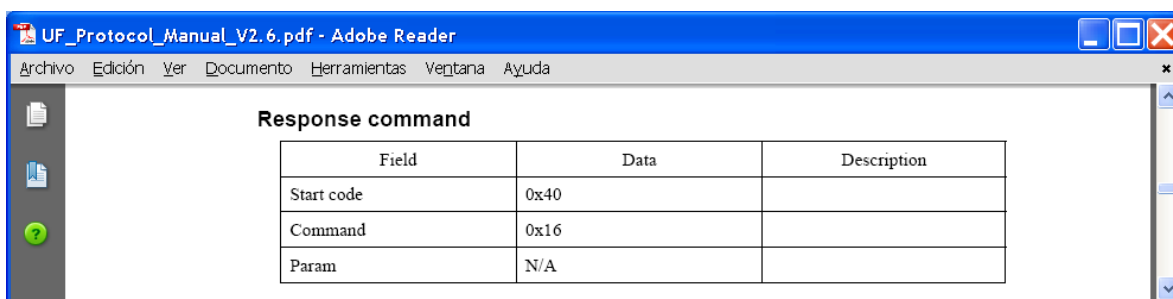
Las figuras 5.23, 5.24 y 5.25 muestran la hoja de datos donde se explican los valores que devuelve el módulo para las instrucciones *Verify by Scan* y *Delete Template*, respectivamente.



The screenshot shows a PDF document titled 'UF\_Protocol\_Manual\_V2.6.pdf' in Adobe Reader. The table 'Response command' is displayed, detailing the fields and data returned by the 'Verify by Scan' instruction.

Field	Data	Description
Start code	0x40	
Command	0x08	
Param	User ID	
Size	Sub-ID	Order of fingerprints enrolled in the same ID
Error	( 0x62 )	( SCAN_SUCCESS )
	0x63	SCAN_FAIL
	0x6C	TIME_OUT
	0x61	SUCCESS
	0x6B	TRY_AGAIN
	0x69	NOT_FOUND
	0x6A	NOT_MATCH

Figura 5.23 Documentación de la instrucción *Verify by Scan*. Tomado del documento *UF\_Protocol\_Manual\_V2.6.pdf* publicado por la empresa Suprema®



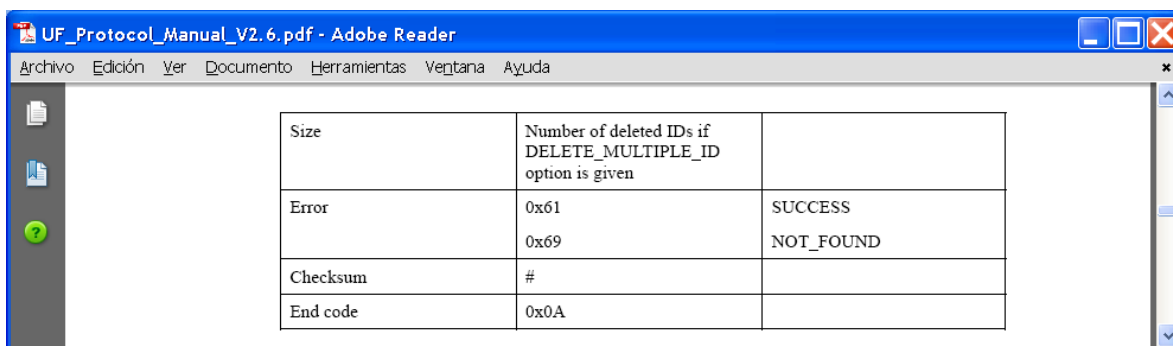
The screenshot shows the same PDF document in Adobe Reader, but displaying the 'Response command' table for the 'Delete Template' instruction.

Field	Data	Description
Start code	0x40	
Command	0x16	
Param	N/A	

Figura 5.24 Documentación de la instrucción *Delete Template*. Tomado del documento *UF\_Protocol\_Manual\_V2.6.pdf* publicado por la empresa Suprema®

Así es como se puede saber cuales son los campos del paquete de datos donde el módulo devuelve su respuesta para cada instrucción. Hay un caso especial, en el cual se envía más información que un solo paquete de datos. Se trata de

la instrucción mediante la cual se obtiene una o más huellas digitales almacenadas dentro del módulo para ser grabadas en un archivo de disco.



Size	Number of deleted IDs if DELETE_MULTIPLE_ID option is given	
Error	0x61 0x69	SUCCESS NOT_FOUND
Checksum	#	
End code	0x0A	

Figura 5.25 Documentación de la instrucción Delete Template (continuación). Tomado del documento UF\_Protocol\_Manual\_V2.6.pdf publicado por la empresa Suprema®

El pseudo código correspondiente a este caso especial se muestra enseguida:

```

Case READ_TEMPLATE

    Select Case pac.Flag_Error

        Case SUCCESS

            <Esperar a que lleguen todos los bytes>

            <Grabar los bytes en un archivo de disco>

        Case [CONTINUE]

            <Esperar a que lleguen todos los bytes>

            <Grabar los bytes en un archivo de disco>
            <Esperar a que lleguen los bytes del encabezado del
            segundo paquete>

            <Obtiene los bytes del encabezado del segundo paquete>

        Select Case pac.Flag_Error

            Case SUCCESS

                <Esperar a que lleguen todos los bytes>

                <Grabar los bytes en un archivo de disco>

            Case [CONTINUE]

                <Esperar a que lleguen todos los bytes>

                <Grabar los bytes en un archivo de disco>

```

```

                                <Vaciar el buffer de recepción>

                                Case Else ' Error

                                resp = Estatus.ERROR

                                End Select

                                Case Else ' Error

                                resp = Estatus.ERROR

                                End Select

```

Ver Anexo B.5

Cuando se solicita al módulo que se envíen las huellas digitales que se han almacenado de un empleado, éste enviará cada una de las huellas junto con un respectivo paquete de datos. De forma que primero llegará un paquete para la primera huella, después llegarán los bytes que conforman la primera huella. Después llegará un paquete para la segunda huella, y después llegarán los bytes que conforman la segunda huella, y así sucesivamente.

Normalmente el resultado de una instrucción como ésta podría recibirse de forma iterativa, por ejemplo, con un bucle *For* o con un bucle *While*. Sin embargo, como solamente se almacenará un máximo de dos huellas por empleado, no hay problema en utilizar dos instrucciones *Select Case*, una para cada huella.

De acuerdo a la hoja de datos, el paquete que acompaña a cada huella digital contendrá uno de tres posibles valores: “Éxito”, “Continuación”, o algún código de error.

Si es “Éxito”, significa que es la última huella digital que el módulo tenía almacenada para ese empleado. Si es “Continuación”, significa que después de ésta huella digital aún viene al menos una mas. Si es “Error”, significa que ha ocurrido alguno de los errores aplicables para esta instrucción, y cualquier dato que se trate de leer después del paquete puede ser también erróneo.

De hecho esto es lo que se hace con esta combinación de instrucciones *Select Case*: Se determina cuál de los tres posibles valores es el que contiene el paquete en cuestión y en base a eso se decide si debe esperar una huella más o si ya es todo.



Este fragmento del código utiliza dos rutinas auxiliares: una para tomar todos los bytes que conforman una huella digital y grabarlos en un archivo de disco, y la otra para vaciar el buffer de recepción de datos, de forma que si se produjo algún error, los bytes que se quedaron en el buffer o que aún vienen en camino sean recibidos e ignorados.

A continuación se muestra la primera de estas dos rutinas:

```
Private Function LeerYGrabarLosBytesDeUnaHuella(ByVal numHuella As Byte)
                                                    As Estatus
    <Iniciar bloque Try-Catch de control de errores>

        <Leer los bytes recibidos en el puerto serie y almacenarlos
        en un arreglo de bytes>

        <Grabar los bytes recibidos en un archivo de disco>

    <Terminar bloque Try-Catch de control de errores>

    <Leer código de terminación (END CODE)>

    <Devuelve Éxito ó Error como resultado de la función>

End Function
```

Ver Anexo B.6

Esta rutina recibe como parámetro el número de huella digital del que se trata (“0” ó “1”) y graba los bytes que ya se encuentran en el buffer de recepción del puerto serie en un archivo cuyo nombre es conformado por el nombre del empleado y el número de huella. Finalmente lee un byte que corresponde con el *END\_CODE* del envío.

La otra rutina se muestra a continuación:

```
Private Sub VaciarBufferDeRecepcion()

    Dim cant As Long = 0
    Dim basura() As Byte

    ' Lee todo lo que haya para que deje de generar el evento
    Do While puerto.BytesToRead > 0

        Delay(1000) ' delay para esperar a que llegue todo
        cant = puerto.BytesToRead
        ReDim basura(cant)
        puerto.Read(basura, 0, cant)

    Loop
End Sub
```

Como se comentó anteriormente, su único propósito es vaciar el buffer de recepción para eliminar de ahí cualquier byte que pudiera generar de nuevo el evento de recepción de datos produciendo que la Terminal desperdicie tiempo tratando de procesar datos inválidos.

Finalmente, dentro del manejador del evento de recepción de datos se cambia el valor de la variable miembro de control *estado*, para que la rutina que mandó la respuesta pueda continuar con su ejecución. El fragmento de código correspondiente al final del manejador del evento se muestra enseguida:

```
End Select

xFin:  ' Termina la rutina
      If Me.errorBytes Then
          ' En caso de error lee todo lo que haya en el buffer y
          ' ejecuta el reset
          Me.VaciarBufferDeRecepcion()
          Me.procesandoRecepcion = False
          Me.Reset()
      End If

      comando = NINGUNO
      _estatus = EstatusLector.Normal
      Me.procesandoRecepcion = False

End Sub
```

Aquí primero se determina si ocurrió algún error durante el proceso de recepción de los datos. Si es así se vacía el buffer de recepción y se ejecuta la instrucción de calibración del equipo para disminuir el riesgo de otros posibles errores. Finalmente se ajustan las variables miembro necesarias para poder continuar con el flujo normal de la aplicación.

En resumen, el proceso de ordenarle una instrucción al módulo lector de uellas digitales se puede resumir en el diagrama de flujo de la figura 5.26.

Cabe aclarar que el modelo SFM3500 de suprema soporta la modalidad de identificación automática. Esto significa que no es necesario investigar si el empleado colocó su huella en el sensor para después ejecutar la instrucción de identificación, sino que cuando el empleado coloca su huella en el sensor, éste realiza la identificación de forma automática y envía inmediatamente su respuesta a la Terminal.

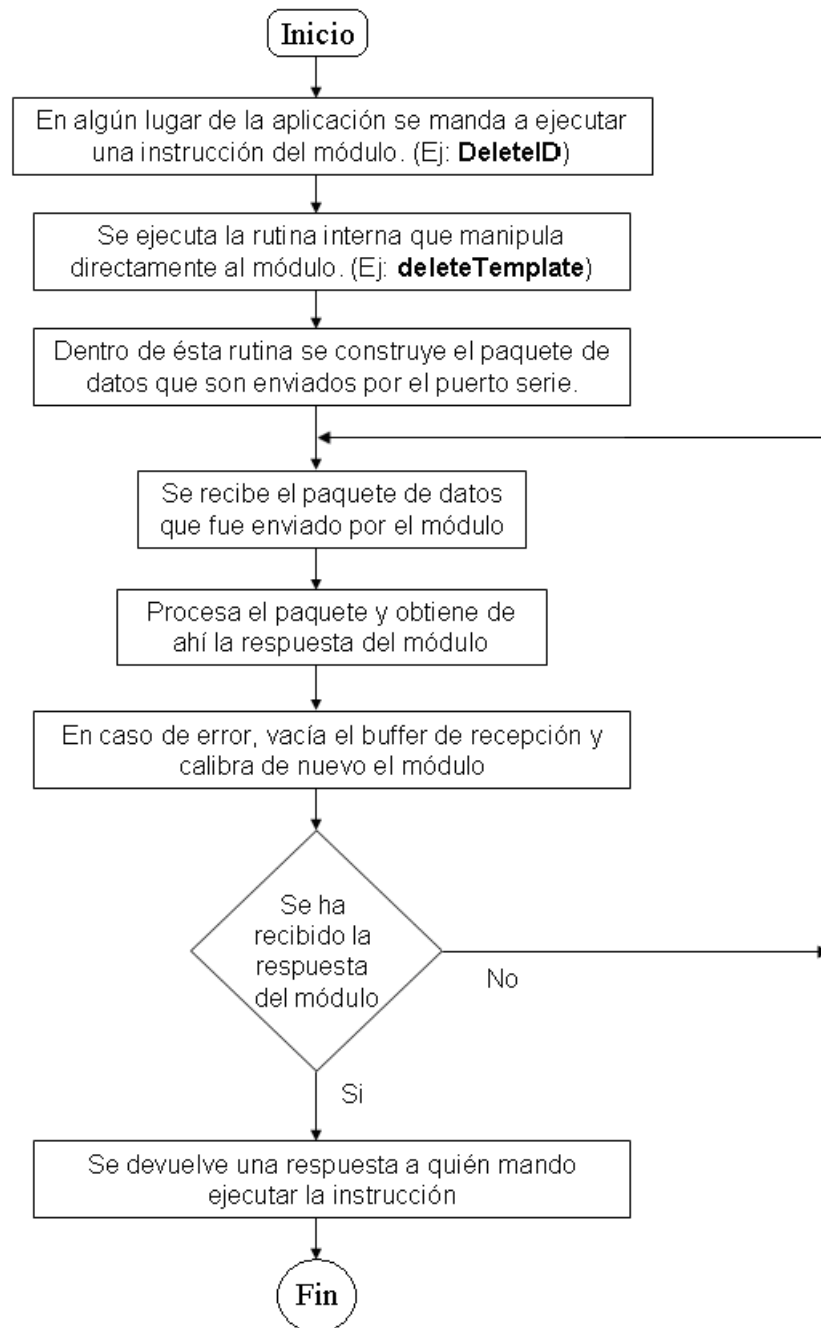


Figura 5.26 *Diagrama de flujo de una petición al módulo lector de huellas digitales*

### 5.6.9 La Clase que Habilita y Deshabilita el Hardware: Hardware

La última clase relacionada con el Hardware es una clase que representa precisamente al Hardware general de la Terminal, y es por ese motivo que se nombró así. La intención de esta clase es englobar todos los elementos de hardware con los que cuenta la Terminal para cuando se requiera ejecutar una

misma instrucción para todos ellos. Por el momento, las únicas instrucciones que son comunes a todos los elementos del hardware de la Terminal son las de *Activar* y *Desactivar*. Esto es, activar/desactivar la cámara fotográfica, o el lector de huellas digitales, o los relevadores, etc.

La función de esta clase se limita entonces a activar y desactivar los diferentes elementos del hardware de la Terminal, dependiendo de los valores indicados en algunas variables de la clase *Config*, de la cual se hablará más adelante. Dentro de la clase *Config* hay una variable que indica el estado que debe tener cada elemento del hardware.

Por lo tanto, basándose en esas variables la clase *Hardware* puede realizar su función. A continuación se muestra el seudo código de esta clase, donde solamente se ejecutan los métodos *Activar* y *Desactivar* de cada una de las clases involucradas con el hardware de la Terminal, y en algunos casos se configura el componente de hardware cuando así se requiera.

```
' Huella Digital
If Config.usaHuellas Then

    If lectorHuella.estatusHardware = Inactivo Then

        Select Case sensor

            Case SensorHuellaDigital.BioScript_MV1510

                <Asignar parámetros al módulo>

                <Activar el módulo>

            Case SensorHuellaDigital.Suprema_SFM3500

                <Activar el módulo>

                <Asignar parámetros al módulo>

        End Select

    End If

Else
    If lectorHuella.estatusHardware = Activo Then
        <Desactivar el módulo>
    End If
End If

' Relevadores
```

```

If Config.usaRelayPermitido Then

    If relPermitir.estatusHardware = Inactivo Then
        <Activar el relevador>
    End If

Else

    If relPermitir.estatusHardware = Activo Then
        <Desactivar el relevador>
    End If

End If

If Config.usaRelayDenegado Then

    If relDenegar.estatusHardware = Inactivo Then
        <Activar el relevador>
    End If

Else

    If relDenegar.estatusHardware = Activo Then
        <Desactivar el relevador>
    End If

End If

' Camara
If Config.sacarFotos Then

    If camara.estatusHardware = Inactivo Then
        <Activar el la cámara fotográfica>
    End If

Else

    If camara.estatusHardware = Activo Then
        <Desactivar el la cámara fotográfica>
    End If

End If

' LectorCodigo de Barras (No se desactiva para poder usar las
' credenciales de control)
If lectorCodBar.estatusHardware = Inactivo Then
    <Activar el lector de código de barras>
End If

If nCodBarras = CMIRG.Terminar Then
    If lectorCodBar.estatusHardware = Activo Then
        <Desactivar el lector de código de barras>
    End If
End If

```

Ver Anexo B.7

### 5.6.10 La Clase Encargada de la Configuración: Config

Esta clase se encarga de controlar todo lo relacionado con la configuración de la Terminal. Cuando se explicaron las tablas que componen la base de datos del sistema, se mencionó una tabla llamada *ConfigLect*, la cual también existe en la base de datos de la Terminal y es idéntica a la que se encuentra en el servidor.

Es mediante los datos almacenados en esta tabla que la Terminal sabe como debe comportarse. Por ejemplo, mediante estos datos la Terminal conoce su nombre (“AA”, “BB”, etc.), el tipo de proceso que realiza (Puerta, Comedor, Reloj Checador), el hardware que debe tener activado (Lector de código de barras, lector de proximidad, lector de huella digital, cámara fotográfica, etc.), los horarios en los que debe realizar los diferentes procesos que realiza, etc.

Esto es, para que la aplicación funcione necesita forzosamente tener grabado un registro en esta tabla. Dicho registro se obtiene del servidor mediante uno de los métodos de la clase *ConexionWS*.

La clase *Config* se encarga entonces de leer el contenido en la tabla de configuración, y mantiene todos los valores obtenidos en variables públicas de forma que estén disponibles para cualquier elemento del hardware que requiera hacer uso de ellas.

Así, por ejemplo, si una persona coloca su dedo en el sensor de huellas digitales y se realiza una identificación exitosa, éste informa a la Terminal que el empleado “X” está solicitando acceso. Entonces la Terminal, en base a los datos obtenidos de la tabla de configuración determina si es suficiente para otorgar el acceso, o si se requiere otro medio de identificación adicional como la credencial con código de barras o la credencial de proximidad.

Además, una vez que la Terminal decidió si debe permitir el acceso o no, nuevamente utiliza los datos de la tabla de configuración, pero esta vez lo hace para determinar las acciones que debe realizar, como son: tomar una fotografía, activar un relevador, reproducir un sonido, registrar un movimiento, etc.

Los métodos más importantes de la clase *Config* son los que acceden a la tabla de configuración que se encuentra en el servidor (mediante el Servicio Web) y el que obtiene la configuración que ya se encuentra dentro de la misma

Terminal. A continuación se muestra un extracto del método encargado de solicitar al servidor la configuración de la Terminal:

```
<Invocar la rutina del servicio Web para obtener la configuración  
que se encuentra almacenada en el servidor>  
  
<Armar una cadena con la instrucción de SQL para grabar la  
configuración dentro de la terminal>  
  
<Ejecutar la instrucción de SQL para grabar la configuración>
```

Ver Anexo B.8

Como puede observarse, se utiliza el Servicio Web para obtener una estructura que almacena todos los valores de configuración necesarios para que la Terminal pueda operar. Posteriormente esos valores son almacenados en su base de datos interna. De esta forma, si en un futuro la Terminal tuviera que ser reiniciada y no se dispone de una comunicación con el servidor, probablemente por motivos de mantenimiento del mismo, la Terminal ya tendrá suficiente información para poder trabajar.

Lo mismo ocurre con un archivo de texto que contiene los parámetros con los cuales se debe configurar el sensor de huellas digitales. A continuación se muestra la forma en que de nuevo se utiliza el Servicio Web para obtener dicho archivo:

```
' Obtiene el archivo de configuracion del modulo de huellas digitales  
If ws.RecibirArchivoConfigHuellaDigitalDesdeElServidor() = Estatus.ERROR  
Then  
    Throw New Exception("Error al obtener la configuración del módulo  
de huella digital desde el servidor.")  
End If
```

Le corresponde a la clase que manipula el Servicio Web recibir el archivo de configuración del sensor y guardarlo en disco:

```
Public Function RecibirArchivoConfigHuellaDigitalDesdeElServidor()  
As Estatus  
  
    <Iniciar bloque Try-Catch de control de errores>  
  
        <Invoca la rutina correspondiente del servicio Web para  
obtener el archivo como un arreglo de bytes>  
  
        <Si el archivo tiene un tamaño de 1 byte, generar una  
excepción>  
  
        <Guardar los bytes recibidos en un archivo de disco>
```

```
<Terminar bloque Try-Catch de control de errores>
```

```
<Devuelve Éxito ó Error como resultado de la función>
```

```
End Function
```

Ver Anexo B.9

De la misma forma se reciben los archivos de sonido que en algún momento serán reproducidos y el logo de la empresa correspondiente (Fuller Cosmetic's en este caso).

Una vez que la Terminal cuenta con toda la información de configuración necesaria, ya puede inicializar la aplicación por si misma, sin necesidad de que exista comunicación con el servidor. Para esto simplemente tiene que leer el contenido tanto de la tabla *ConfigLect* como del archivo de configuración del sensor, y almacenar los valores en variables globales o estáticas.

Además de todo esto, hay un detalle que se debe tomar en consideración: cuando la Terminal ejecuta la aplicación por primera vez, no tiene aún la configuración necesaria para poder trabajar, así que debe solicitarla al servidor. Pero es posible que haya más de una Terminal en el sistema y, además, cada una realizará una labor diferente (controlar una puerta, el acceso al comedor, etc.). Así que desde un principio cada Terminal deberá tener una configuración diferente.

Esto significa que para que una Terminal pueda solicitar al servidor su información de configuración, debe conocer su nombre mediante el cual se identificará ante el servidor. Para esto se implementó una rutina que solicita al usuario administrador del sistema introduzca el nombre de la Terminal en cuestión. Una vez que se le ha asignado un nombre, la Terminal lo graba en la tabla *ConfigLect* y posteriormente lo utiliza para obtener la configuración vía el Servicio Web.

#### **5.6.11 La Clase que Representa a la Terminal como un Objeto: CLectora**

Esta clase contiene algunas variables miembro y algunos métodos que representan a la Terminal desde el punto de vista de la programación, esto es, vista como un objeto de software. Se puede decir que esta clase complementa a la clase *Hardware* descrita anteriormente. A continuación se muestran las variables miembro que contiene esta clase:



```

Public lectora As String = ""

Public _estatus As EstatusLectora = EstatusLectora.SinActividad

Public estadoAdmin As EstatusAdmin = EstatusAdmin.EsperandoCredencial

Public tiempoSinActividad As Long = 0

Public tiempoCalibracion As Long = 0

Public siguienteComunicacion As ComunicacionLectora =
    ComunicacionLectora.Inicio

Public terminoComunicacion As Boolean = True

```

La variable *lectora* contendrá el nombre de la lectora, por ejemplo “AA” o “BB”. Este se utiliza en varias de las rutinas del Servicio Web. Las variables *\_estatus* y *estadoAdmin* se utilizan para conocer y controlar el estado actual en el que se encuentra la lectora. Estos son los posibles valores que puede tomar cada una:

<b>Public Enum EstatusLectora As Byte</b>	<b>Public Enum EstatusAdmin As Byte</b>
SinActividad	EsperandoCredencial
Trabajando	ConfirmandoNumero
ComunicandoseWS	EsperandoHuella1
AdminHD	EsperandoHuella2
AdminManual	EnviandoHuellas
<b>End Enum</b>	<b>End Enum</b>

Las variables *tiempoSinActividad* y *tiempoCalibracion* se utilizan para contabilizar el tiempo que ha transcurrido desde la última vez en que la Terminal realizó algún proceso, y en base a esto decidir si es momento de comunicarse con el servidor para enviar y recibir datos, o si es momento de recalibrar el sensor de huellas digitales, o si simplemente debe seguir esperando para realizar alguna acción.

Para esto, en el programa principal se definió un temporizador, de forma que cada que ha transcurrido cierta cantidad de tiempo éste se suma en las variables antes mencionadas invocando los siguientes métodos:

```

Public Sub SumarTiempoSinActividad(ByVal tiempo As Long)

    tiempoSinActividad += tiempo

End Sub

Public Sub SumarTiempoDeCalibracion(ByVal tiempo As Long)

```

```
Me.tiempoCalibracion += tiempo  
  
End Sub
```

Las últimas dos variables, *siguienteComunicacion* y *terminoComunicacion* se usan para controlar la comunicación con el servidor vía el Servicio Web.

Otro método relevante de esta clase que vale la pena mencionar es el siguiente:

```
Public Sub DarAcceso(ByRef form As frmInicio,  
                    ByVal acceso As Boolean)  
  
    If acceso Then  
        ' Se permitio el acceso  
  
        CMensajes.AsignarMensaje(form, Config.mensajePermitido)  
  
        sonidos.Bien()  
  
        If Config.usaRelayPermitido Then relPermitir.Cerrar(form)  
  
    Else  
        ' Se denego el acceso  
  
        CMensajes.AsignarMensaje(form, Config.mensajeDenegado)  
  
        sonidos.Mal()  
  
        If Config.usaRelayDenegado Then relDenegar.Cerrar(form)  
  
    End If  
  
End Sub
```

Este método es el encargado de realizar las acciones necesarias cuando un usuario solicita acceso colocando su dedo en el lector de huellas digitales. Una vez que el módulo ha identificado a la persona o ha decidido que ésta no existe, y que la Terminal ha realizado las verificaciones necesarias para conceder o denegar el acceso, se ejecuta esta rutina en la cual:

1. Se actualizan los mensajes en pantalla, por ejemplo, para mostrar: “Adelante”, “Acceso Denegado”, etc.
2. Se ejecuta el sonido adecuado, que puede ser incluso una voz indicando que puede o no puede pasar.
3. Se activan los relevadores en caso de que la configuración de la Terminal así lo indique.

De forma tal que en el programa principal basta con invocar ésta rutina para que el usuario conozca la respuesta a su petición.

#### **5.6.12 Las Clases de Negocio CPersonal y CMovimientos**

Estas clases se encargan de ejecutar las rutinas del Servicio Web utilizando para ello a la clase *ConexionWS* que se describió anteriormente y registran en la base de datos de la Terminal los datos relacionados con el personal de la empresa y con los movimientos que han ocurrido, respectivamente.

Así, la clase *CPersonal* es la encargada de actualizar la tabla de *Personal* tanto para agregar a los nuevos empleados, como para borrar los registros que aquellos empleados que ya fueron dados de baja. También se encarga de enviar al servidor las huellas digitales que se capturaron en su propio módulo y de solicitarle aquellas que fueron capturadas en otras terminales.

Por otra parte, la clase *CMovimientos* se encarga de grabar dentro de la base de datos de la Terminal los registros correspondientes a los accesos que va teniendo cada empleado en dicha Terminal, esto es, graba las horas de entrada y de salida de los empleados y las horas de entrada y de salida del comedor, dependiendo del tipo de Terminal que se trate. También se encarga de enviar esos movimientos al servidor, así como de eliminarlos varios días después de haber sido enviados. Por último, es también la encargada de enviar las fotografías que hayan sido capturadas al servidor en el caso de que la configuración de la Terminal así lo indique.

Para realizar todos los procesos relacionados con el envío y recepción de datos al servidor se utilizan los métodos de la clase *ConexionWS*, la cual directamente invoca los métodos del Servicio Web.

#### **5.6.13 La Clase Encargada de Hacer las Verificaciones: CVerificador**

La función de esta clase es, como su nombre lo indica, verificar la existencia de un empleado en la base de datos de la Terminal. Si el empleado efectivamente existe, entonces verifica su registro para saber, de acuerdo a los valores almacenados en los campos de *fechaIngreso* y *fechaBaja*, si se le debe permitir o no el acceso solicitado.

En el caso de que la Terminal esté configurada para trabajar como control de acceso al comedor, la clase *CVerificador* consulta la cantidad de veces que ese

empleado ha entrado a comer y/o a desayunar, y en base a eso determinar si ya ha agotado sus desayunos y/o comidas disponibles para ese día.

Claro está que el trabajo de esta clase comienza a partir de que un empleado ha colocado su dedo en el sensor de huellas digitales y el módulo lo ha identificado satisfactoriamente y ha avisado a la Terminal de quién se trata. En los casos en que el módulo no logre hacer una identificación satisfactoria por tratarse de una persona ajena a la empresa o al área correspondiente de la misma, el módulo simplemente informará a la Terminal de que alguien no autorizado solicitó acceso, para que ésta ejecute las acciones correspondientes a un acceso denegado, y entonces la clase *CVerificador* no tiene que hacer absolutamente nada, pues no hay nadie de quien se deba verificar su acceso.

Esta clase no utiliza el Servicio Web. Todo su proceso consiste en realizar consultas a la base de datos de la Terminal.

#### **5.6.14 La Clase Encargada de los Mensajes: CMensajes**

El propósito de esta clase es controlar y manipular todos los mensajes que la Terminal muestra en la pantalla. Existen diferentes tipos de mensajes, y esta clase los controla todos.

En general se dispone de seis tipos de mensajes:

Uno es el mensaje que aparece siempre que la Terminal esté encendida, dando la indicación de que se debe colocar el dedo en el sensor de huellas digitales, o tal vez simplemente saludando a quien esté ahí presente. Este mensaje incluye un pequeño título donde se puede colocar el nombre de la empresa (por ejemplo: “Fuller”).

Otro tipo de mensaje es el que aparece cuando un usuario ha solicitado acceso y la Terminal se lo ha concedido. Es un mensaje de tipo: “Adelante Gabriel Ibarra” o “Bienvenido, Gabriel I R”. El mensaje que complementa a éste es el que aparece cuando la Terminal deniega el acceso. Un ejemplo de éste tipo de mensaje sería: “Acceso Denegado” o “No tiene autorización para ingresar”.

Además de éstos, se añadieron dos mensajes de control, que sirven para saber lo que la Terminal está procesando en un determinado momento. Uno de ellos es especialmente para que la persona que está interactuando con la Terminal tenga conocimiento de lo que ésta hace. Por ejemplo, en el caso de la Terminal

que está configurada como Reloj Checador el mensaje será precisamente: “Reloj Checador”. Si está en modo de pruebas, el mensaje será: “Modo de Pruebas”. El otro mensaje está pensado para que los administradores puedan dar una pista al responsable del correcto funcionamiento del sistema cuando éste llega a fallar, es un mensaje que indica exactamente el proceso que está realizando en cada instante la Terminal, por ejemplo: “Actualizando personal mediante servicio web” o “Enviando movimientos al servidor mediante servicio web”.

El último tipo de mensaje simplemente muestra la fecha y hora actual. Además de esto, algunos mensajes se pueden personalizar un poco mediante *tags* especiales. Por ejemplo, si se requiere que a un empleado se le felicite por ser su cumpleaños, le aparecerá un mensaje similar al siguiente:

*Bienvenido*  
*12234 Gabriel Ibarra*  
*Feliz Cumpleaños*

El mensaje en la tabla de configuración debe construirse de la siguiente forma:

*Bienvenido*  
*#4 #3*  
*#5*

Donde #3 representa el nombre del empleado, esto es, cuando la clase construye el mensaje debe reemplazar la combinación #4 con el nombre del empleado correspondiente. Lo mismo ocurre con el número de empleado (#4) y con el mensaje de “Feliz Cumpleaños” (#5).

A continuación se muestra un fragmento del código que prepara el mensaje para mostrar el nombre del empleado:

```
' Formatea el nombre
cad = CVerificador.nombre
mensaje = Replace(mensaje, "#3", cad)
```

De esta misma forma se van formando los diferentes mensajes que muestra la Terminal. Por último, simplemente se asignan los mensajes a la propiedad *Text* de cada una de los controles de tipo *Label* que posee la pantalla.

### 5.6.15 La Clase CMsgBox

Esta clase trabaja en combinación con una de las pantallas de la aplicación. Su propósito es hacer las veces de un cuadro de dialogo con tres funcionalidades: realizar una pregunta al usuario y solicitar una respuesta de tipo “Si” o “No”, solicitar al usuario que introduzca información en forma de texto, y mandar un mensaje informativo.

La intención de utilizar esta clase es que en el caso del mensaje informativo, éste debía desaparecer automáticamente después de dos o tres segundos, comportamiento que ninguna de las instrucciones nativas de Visual Basic posee.

Además, al ser un cuadro de diálogo creado por uno mismo, se puede personalizar en cuanto a colores y diseño, ayudando así a lograr una vista un poco más amigable para el usuario.

Esta clase trabaja en conjunto con tres de las pantallas que conforman el sistema:

- **frmWaitWindow:** Es la que muestra un mensaje y se oculta automáticamente.
- **frmPreguntaSiNo:** Realiza una pregunta al usuario y espera una respuesta de tipo “Si” o “No”.
- **frmInputBox:** Solicita al usuario alguna información en forma de cadena de texto.

A continuación se muestra el código de la clase *CMsgBox*:

```
Public Class CMsgBox
    Public Shared _mensaje As String = ""
    Public Shared _respuesta As String = ""
    Public Shared _tiempo As Long = 0
    Public Shared _RespSiNo As Respuesta = Respuesta.No

    Public Shared Sub InputBox(ByVal mensaje As String)
        _mensaje = mensaje

        ' Usa un form para mandar el mensaje
        Dim ww As New frmInputBox
        ww.ShowDialog()
    End Sub

    Public Shared Sub WaitWindow(ByVal mensaje As String,
                                Optional ByVal tiempo As Long = 2000)
```

```

        _mensaje = mensaje
        _tiempo = tiempo

        ' Usa un form para mandar el mensaje
        Dim ww As New frmWaitWindow
        ww.ShowDialog()
    End Sub

    Public Shared Sub PreguntaSiNo(ByVal mensaje As String)
        _mensaje = mensaje
        ' Usa un form para mandar el mensaje
        Dim ww As New frmPreguntaSiNo
        ww.ShowDialog()
    End Sub
End Class

```

Contiene cuatro variables miembro de tipo público: *\_mensaje*, *\_respuesta*, *\_RespSiNo* y *\_tiempo*. La primera se utiliza para almacenar ahí el mensaje que se mostrará en el cuadro de diálogo. La segunda contendrá la respuesta del usuario una vez que el cuadro de diálogo se haya cerrado. La tercera contendrá la respuesta del usuario en el caso de la pantalla *frmPreguntaSiNo*. La última la utilizará la pantalla *frmWaitWindow* para saber después de cuanto tiempo debe cerrarse.

Las tres rutinas hacen básicamente lo mismo: asignan en la variable miembro el texto del mensaje que recibieron como parámetro y ejecutan la pantalla correspondiente.

El código en cada una de estas pantallas es muy sencillo, y se explicará junto con la imagen de cada una.

### Pantalla **frmWaitWindow**:

Al abrir la pantalla se asigna el texto del mensaje que se debe mostrar en un control *Label* y asigna el tiempo que debe transcurrir antes de cerrar la pantalla en un control *Timer*. Cuando el control *Timer* dispara su evento, la pantalla simplemente se cierra.

```

Public Class frmWaitWindow

    Private Sub frmWaitWindow_Load(ByVal sender As System.Object,
                                    ByVal e As System.EventArgs) Handles MyBase.Load

        Me.lblMensaje.Text = CMsgBox._mensaje
        Me.Timer.Interval = CMsgBox._tiempo
        Me.Timer.Enabled = True
    End Sub
End Class

```

```

End Sub

Private Sub Timer_Tick(ByVal sender As System.Object,
                      ByVal e As System.EventArgs) Handles Timer.Tick
    Me.Close()
End Sub

End Class

```

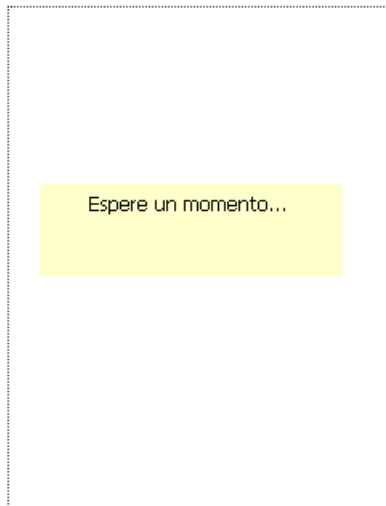


Figura 5.27 Mensaje generado por la clase CMsgBox

### Pantalla **frmPreguntaSiNo**:

De igual forma, al abrir la pantalla se asigna el texto del mensaje que se debe mostrar en un control *Label*. Solo que esta pantalla se cierra mediante un botón y, al hacerlo, se asigna la respuesta seleccionada a la variable miembro que la almacenará.

```

Public Class frmPreguntaSiNo

    Private Sub cmdSi_Click(ByVal sender As System.Object,
                          ByVal e As System.EventArgs) Handles cmdSi.Click
        CMsgBox._RespSiNo = Respuesta.Si
        Me.Close()
    End Sub

    Private Sub cmdNo_Click(ByVal sender As System.Object,
                          ByVal e As System.EventArgs) Handles cmdNo.Click
        CMsgBox._RespSiNo = Respuesta.No
        Me.Close()
    End Sub

End Class

```



```

Private Sub frmPreguntaSiNo_Load(ByVal sender As System.Object,
                                ByVal e As System.EventArgs) Handles MyBase.Load
    Me.lblMensaje.Text = CMsgBox._mensaje
End Sub

End Class

```

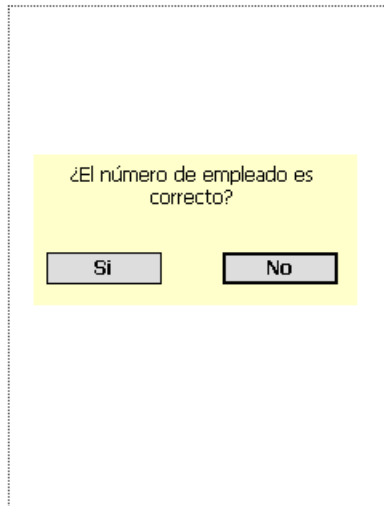


Figura 5.28 *Pregunta generada por la clase CMsgBox*

### Pantalla **frmInputDialog**:

Es idéntica a la anterior, solo que al cerrar en vez de almacenar una respuesta que esté limitada a “Si” o “No”, almacena la cadena de texto introducida por el usuario.

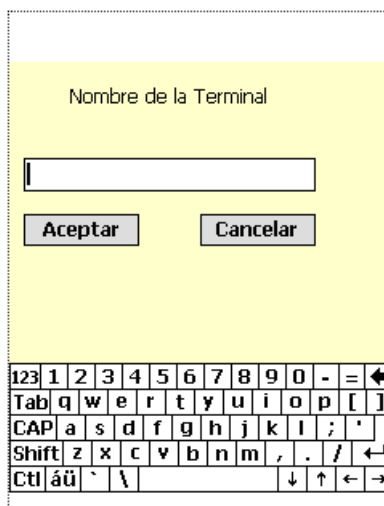


Figura 5.29 *Entrada del usuario solicitada por la clase CMsgBox*

A continuación se explicará el funcionamiento de las dos pantallas principales de la aplicación. La primera es la pantalla que siempre estará activa y es la que procesará el registro de los empleados. La segunda es la pantalla que permite capturar la huella de un empleado de nuevo ingreso.

#### 5.6.16 La Pantalla frmInicio

Esta es la pantalla principal de la aplicación. Es la que aparece constantemente mientras la Terminal este encendida. En ella se muestran los mensajes que la Terminal manda al usuario y una vez que se ha ejecutado, la única forma de control posible es mediante las credenciales de control.

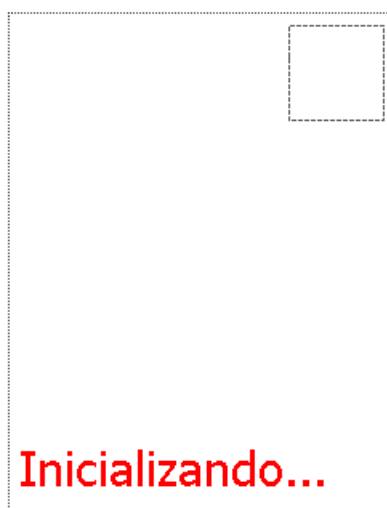


Figura 5.30 *Pantalla principal de la aplicación*

Las credenciales de control son un conjunto de siete credenciales creadas especialmente para manipular la Terminal. El contenido de las mismas es un código de barras que representa una palabra clave para aplicación. A continuación se enlistan las palabras clave de cada una de esas credenciales:

- **CMIRG Terminar:** Cierra la aplicación o la pantalla de captura de huellas digitales.
- **CMIRG AdminHD:** Abre la pantalla de captura de huellas digitales.
- **CMIRG Comidas:** Establece la Terminal del comedor en modo *comidas*.
- **CMIRG Desayunos:** Establece la Terminal del comedor en modo *desayunos*.

- **CMIRG Default:** Establece la Terminal del comedor en modo normal, esto es, será comidas o desayunos dependiendo de la hora y de los datos grabados en la tabla de configuración.
- **CMIRG IniPractica:** Inicia el modo de práctica.
- **CMIRG FinPractica:** Termina el modo de práctica.

Todas las credenciales llevan el prefijo “CMIRG” que es el nombre de la empresa encargada del desarrollo del sistema. Se trabajó así para que no exista nunca la posibilidad de que alguna persona no autorizada consiga un código de barras cualquiera y por pura casualidad coincida con una credencial de control. Cuando es necesario realizar alguna acción, los administradores del sistema deslizan la credencial correspondiente y entonces la Terminal responde a lo solicitado.

Para explicar el funcionamiento de la pantalla principal se enlistarán primero las funciones que realiza en orden una por una, y posteriormente serán explicadas. Las funciones son las siguientes:

1. Inicializar la aplicación
2. Realizar una acción respecto a la credencial de control que se deslizó si se deslizó alguna
3. Procesar la huella digital que se colocó en el sensor si se colocó alguna
4. Actualizar los mensajes que se muestran en pantalla
5. Procesar la comunicación con el servidor
6. Volver al paso 2

El proceso de Inicialización realiza los siguientes pasos:

Primero se conecta a la base de datos mediante el método *Conectar* de la clase *BaseDatos*. Después, obtiene el nombre de la Terminal que se encuentra en la tabla de configuración de la misma. Si este nombre no existe, esto es, si la Terminal aún no tiene un nombre, entonces se le solicita al usuario que introduzca uno utilizando la clase *CMsgBox*.

Ya que se cuenta con el nombre de la Terminal, se intenta obtener la configuración desde el servidor mediante el Servicio Web. Si fue posible conectarse al servidor, la configuración se graba en la base de datos. Si no, entonces no pasa nada.

Una vez que se tiene la configuración actualizada en la base de datos de la Terminal, se obtiene mediante la clase *Config*, que almacena en memoria todos los valores grabados en la tabla de configuración.

Después obtiene, también mediante el Servicio Web, el logo de la empresa y los sonidos que reproducirá la Terminal. Por último, habilita el hardware que se usará en los diferentes procesos (lector de código de barras, lector de huella digital, cámara fotográfica, etc.) utilizando la clase *Hardware*.

Una vez que la Terminal ha inicializado correctamente, la aplicación inicia un ciclo que dura indefinidamente y en el cual se realiza todo el proceso de control de acceso. La rutina principal del proceso es la siguiente:

```
Private Sub Procesar()  
  
    Do  
        Application.DoEvents()  
  
        ProcesarEntradas()  
  
        LimpiarVariables()  
  
        If terminando Then Exit Sub ' Termina la aplicacion  
  
        EjecutarCalibracion()  
  
        ProcesarComunicacion()  
  
        ProcesarMensajes()  
  
    Loop  
  
End Sub
```

Esto es, primero presta atención a los eventos generados en cualquier parte de la aplicación. Es aquí cuando la Terminal puede permitir que el lector de código de barras obtenga el código impreso en una credencial para poder posteriormente trabajar con dicho valor.

Después de esto la Terminal ejecuta una rutina que primero evalúa el valor de la variable *nCodBarras*, la cual contiene cualquier valor de código de barras obtenido por el lector. Esta evaluación se realiza por medio de una instrucción *Select Case*. A continuación se muestra un fragmento de esta evaluación:

```
Select Case nCodBarras  
    Case CMIRG.Terminar
```

```

        CMensajes.AsignarMensajeEspecial(Me, "Terminando...",
                                           False)

        camara.TomarFoto(0)
        CMovimientos.RegistrarMovimiento()
        Terminar()
        Exit Sub

    Case CMIRG.PracticaFin
        CMensajes.AsignarMensajePequeño(Me, "Pasaron credencial
                                           Fin Practica")

        objLectora.estatus = EstatusLectora.Trabajando
        modoPractica = False
        CMensajes.AsignarMensajeEspecial(Me, "Termina Práctica")
        hizoAlgo = True
        GoTo xFin

    Case CMIRG.PracticaIni
        CMensajes.AsignarMensajePequeño(Me, "Pasaron credencial
                                           Inicio Practica")

        objLectora.estatus = EstatusLectora.Trabajando
        modoPractica = True
        CMensajes.AsignarMensajeEspecial(Me, "Modo de Práctica")
        hizoAlgo = True
        GoTo xFin

```

Como puede observarse, se examina el valor del código de barras obtenido y en base a esto se realizan las acciones correspondientes.

Así es como son procesadas las credenciales de control. En el caso de las huellas digitales la rutina realiza un proceso especial, parte del cual se muestra a continuación:

```

If Config.usaHuellas Then
    Select Case sensor

        Case SensorHuellaDigital.Suprema_SFM3500
            ' Ve si pusieron el dedo
            If Not nHuellaId = "" Then
                objLectora.estatus = EstatusLectora.Trabajando
                CMensajes.AsignarMensajePequeño(Me, "Pusieron
                                                    huella digital")

                ' Hace las verificaciones necesarias
                If CVerificador.ExisteEmpleado(, , nHuellaId) Then
                    acceso = CVerificador.VerificarComidas()
                End If

                RegistrarMovimiento(acceso, "H")
                hizoAlgo = True
                GoTo xFin
            End If
        End Select
    End If

```

El sensor realiza la identificación de la persona de forma automática y asigna el valor “0” a la variable *nHuellaId* si la identificación no fue exitosa y el número del empleado si sí lo fue.

Luego entonces, si la variable tiene algún valor distinto de una cadena vacía, muestra primero un mensaje para indicar que va a procesar una verificación de huella digital. Posteriormente utiliza la clase *CVerificador* para determinar si el empleado tiene acceso o no. Si no lo tiene, entonces se producirá el mismo efecto que si el sensor hubiera devuelto el valor “0”.

Una vez tomada la decisión de si el empleado debe pasar o no, se ejecuta una rutina encargada de procesar el acceso, invocando al método *DarAcceso* de la clase *CLectora* para proporcionar una respuesta al usuario, grabando el movimiento en base de datos mediante la clase *CMovimientos*, capturando una fotografía si es necesario y realizando después una pausa para darle tiempo al usuario de que perciba la respuesta de la Terminal.

Este es el proceso principal donde la Terminal controla el acceso de los empleados. Después de limpiar algunas variables de control, actualizar los mensajes que se muestran en pantalla, etc., la Terminal ejecuta la rutina de comunicación con el servidor. Esta rutina se basa en la siguiente enumeración:

```
Public Enum ComunicacionLectora As Byte
    Inicio
    Recibir_CambiosPersonal
    Recibir_Personal
    Enviar_Movimientos
    Borrar_Movimientos_Personal
    Enviar_Fotos
    Enviar_Huellas
    Recibir_CambiosHuellas
    Recibir_Huellas
    Fin
End Enum
```

De forma que mediante una variable de control a la cual se le van asignando todos y cada uno de estos posibles valores la Terminal va procesando cada una de las funciones que proporciona el Servicio Web. En el siguiente fragmento de pseudo código se muestra la forma en que lo hace:

```
Select Case objLectora.siguienteComunicacion

    Case ComunicacionLectora.Inicio

        <Mostrar mensaje de estatus>
```

```

        <Incrementar la variable de control>

    Case ComunicacionLectora.Recibir_CambiosPersonal

        <Mostrar mensaje de estatus>

        <Recibir el listado de personas que se deben obtener
        del servidor, separadas por comas, mediante los métodos
        de la clase CPersonal>

        <Incrementar la variable de control>

    Case ComunicacionLectora.Recibir_Personal

        <Mostrar mensaje de estatus>

        <Si el listado se encuentra vacío, incrementar la
        variable de control>

        <Si el listado no se encuentra vacío, obtener los datos
        de la siguiente persona y quitarla del listado>

    Case ComunicacionLectora.Envia_Movimientos

        <Mostrar mensaje de estatus>

        <Enviar los movimientos al servidor en bloques de 10 en
        10, mediante los métodos de la clase CMovimientos>

        <Si ya no hay más movimientos que enviar, incrementar
        la variable de control>

```

**Ver Anexo B.10**

En este pseudo código puede observarse cómo simplemente se van ejecutando los diferentes métodos del Servicio Web mediante las clases correspondientes (*CPersonal*, *CMovimientos*, etc.) y se va incrementando el valor de una variable de control. Si se detecta una falla de comunicación con el Servicio Web debida probablemente a una desconexión del cable de red, o a un servicio de mantenimiento en el servidor, se devuelve el valor de la variable de control a su valor original, para así no seguir desperdiciando tiempo tratando de establecer una comunicación que temporalmente será imposible de lograr y mejor utilizar esa capacidad de proceso para las funciones de control de acceso.

Finalmente, cuando se desliza la credencial de control etiquetada como “Terminar”, la aplicación termina el ciclo indefinido de ejecución y antes de cerrar la pantalla ejecuta la siguiente subrutina donde se desactiva todo el hardware de la Terminal:

```

Private Sub Terminar()

    CMensajes.AsignarMensajePequeño(Me, "Desactivando hardware")

    terminando = True
    TimerInactividad.Enabled = False
    TimerRelays.Enabled = False

    ' Se desconecta de la base de datos
    bd.Desconectar()

    ' Con esto desactiva todo el hardware
    Config.usaBarras = False
    Config.usaHuellas = False
    Config.usaProximidad = False
    Config.sacarFotos = False
    Config.usaRelayPermitido = False
    Config.usaRelayDenegado = False

    Hardware.Actualizar()

End Sub

```

El resto del código en la pantalla principal consiste en los manejadores de los eventos de los *Timers*, en los cuales se va acumulando el tiempo transcurrido desde la última vez que la Terminal procesó algo. Esto con el fin de que no se realice comunicación con el servidor en las horas pico cuando todos los empleados desean checar su entrada. En vez de eso se espera hasta que haya pasado el tiempo establecido en la configuración de la Terminal sin que nadie haya checado.

### 5.6.17 La Pantalla frmAdminHD

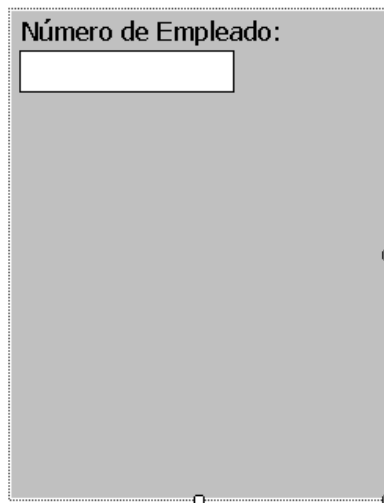


Figura 5.31 *Pantalla de captura de huellas digitales*



En esta pantalla es donde el administrador del sistema captura las huellas digitales de los empleados. Se ingresa a ella deslizando la credencial de control etiquetada como “AdminHD”, y se cierra deslizando la credencial de control etiquetada como “Terminar”.

Al iniciar la pantalla se ejecuta el código de calibración del sensor de huella digital para así lograr mejores resultados:

```
Private Sub frmAdmin_Load(ByVal sender As System.Object,
                          ByVal e As System.EventArgs) Handles MyBase.Load
    lectorHuella.Calibrar()
End Sub
```

Una vez iniciada la pantalla, su funcionamiento se basa en una variable de control llamada *estadoAdmin*, que cambia de valor dependiendo de la última acción realizada. Los posibles valores que puede tomar son los siguientes:

```
Public Enum EstatusAdmin As Byte
    EsperandoCredencial
    ConfirmandoNumero
    EsperandoHuella1
    EsperandoHuella2
    EnviandoHuellas
End Enum
```

Al principio la Terminal está esperando a que se deslice la credencial del empleado en el lector del código de barras. Al hacerlo, la Terminal recibe el número del empleado impreso como código de barras y continua con el siguiente paso:

```
Case EstatusAdmin.EsperandoCredencial
    ' Mensaje credencial
    CMensajes.AsignarMensajeAdmin(Me, "Pase Credencial...", False)
    Me.txtEmpleado.Text = ""
    lblNombre.Text = ""
    CVerificador.uiden = 0
```

El encargado de que la variable de *estadoAdmin* incremente su valor en este caso es la misma clase *LectorCodigoBarras*, al leer el código de barras de la credencial. Aquí se muestra el fragmento de código donde se realiza esta acción:

```
If objLectora.estatus = EstatusLectora.AdminHD Then
    objLectora.estadoAdmin = EstatusAdmin.ConfirmandoNumero
End If
```

Después de esto, la Terminal utiliza la clase *CVerificador* para verificar que el empleado exista. Si es así, muestra su nombre y continua con el siguiente paso. Si no, entonces devuelve a la variable de *estadoAdmin* su valor original. Si por el contrario la credencial que se deslizó fue la credencial de control etiquetada como “*Terminar*”, simplemente se cierra la pantalla. A continuación se muestra el pseudo código correspondiente:

**Case** EstatusAdmin.ConfirmandoNumero

```
<Si la credencial que se deslizó fué la de "Terminar",
ejecutar la instrucción Exit Sub>

<Si se deslizó cualquier otra credencial, mostrar mensaje
indicando que la terminal está trabajando>

<Verificar, mediante la clase CVerificador, si el empleado
existe en la base de datos>

<Si el empleado existe>

    <Mostrar su nombre>

    <Borrar del módulo cualquier huella digital que estuviera
ligada a ese número de empleado>

    <Incrementar la variable de control>

<Si el no empleado existe>

    <mostrar un mensaje de aviso y volver a esperar a que se
deslice una credencial>
```

**Ver Anexo B.11**

Como puede observarse, antes de proseguir a grabar las huellas digitales, primero se eliminan las que ese empleado tuviera para que siempre tenga un máximo de dos. Después el valor de la variable de control *estadoAdmin* es incrementado para proseguir con la captura. Entonces se utiliza el método *Enroll* de la clase *LectorHuellaDigital* para obtener las dos huellas digitales del empleado, una por una. El pseudo código correspondiente se muestra enseguida:

**Case** EstatusAdmin.EsperandoHuella1

```
<Mostrar mensaje solicitando al usuario que coloque su huella
digital>

<Capturar la huella digital mediante los métodos de la clase
que controla el módulo>

<Si la huella digital fue obtenida con éxito>
```

```
<Actualizar el campo "huella" de la tabla "personal" para
indicar que ya se han capturado las huellas digitales del
empleado>
```

```
<Mostrar mensaje informando acerca de la calidad con la
que se obtuvo la huella digital>
```

```
<Incrementar la variable de control>
```

```
<Si no fue posible obtener la huella digital con éxito>
```

```
<Mostrar mensaje avisando que ocurrió un error>
```

**Ver Anexo B.12**

Este código sólo obtiene la primera huella digital, y se repite de forma casi idéntica para capturar la segunda. Una vez capturadas ambas huellas, éstas se copian del módulo de huellas digitales al disco duro de la Terminal, y posteriormente se envían al servidor para que otras terminales las puedan bajar y grabar en sus propios módulos. El seudo código es el siguiente:

**Case** EstatusAdmin.EnvioandoHuellas

```
<Mostrar mensaje avisando acerca del envío>
```

```
<Obtiene las huellas digitales recién capturadas y las graba
en archivos de disco mediante los métodos de la clase que
controla el módulo>
```

```
<Realiza el envío de cada una de las huellas digitales
mediante las rutinas del servicio Web>
```

```
<Una vez realizado el envío, elimina los archivos del disco>
```

```
<Si no fue posible hacer el envío, mantiene los archivos de
disco para enviarlos en un futuro>
```

```
<Incrementa la variable de control>
```

**Ver Anexo B.13**

Finalmente se reestablece el valor de la variable de control *estadoAdmin* para proceder a repetir de nuevo todo el ciclo. Todo este proceso es ejecutado dentro de un evento de un control *Timer* que se dispara cada 100ms.

También dentro del mismo evento del *Timer* se permite al empleado que coloque su huella digital. Esto con el fin de que una vez que se han grabado sus huellas, pueda hacer pruebas para saber si éstas realmente se grabaron de forma satisfactoria, o si es conveniente volver a grabarlas. A continuación se muestra el seudo código correspondiente:

*<Si el empleado coloca el dedo en el lector de huellas digitales sin antes haber deslizado su credencial>*

*<Se verifica que el empleado exista mediante la clase CVerificador>*

*<Si el empleado existe>*

*<Se muestra su nombre>*

*<Se reproduce el sonido que está ligado al estatus de "Acceso Permitido">*

*<Si el empleado no existe>*

*<Se muestra una leyenda indicando que el empleado no existe>*

*<Se reproduce el sonido que está ligado al estatus de "Acceso Denegado">*

*<Se ejecuta un retardo de 2 segundos, para que se pueda observar la respuesta>*

*<Se vuelve la pantalla a su estado anterior>*

**Ver Anexo B.14**

Aquí la Terminal verifica si el empleado existe, y en tal caso muestra su nombre en pantalla durante dos segundos utilizando la clase *CMensajes*, para después volver a su estado anterior solicitando que un empleado deslice su credencial. En caso de que la identificación no sea exitosa, la clase *LectorHuellaDigital* devolverá un valor de cero, siendo un número que ningún empleado tendrá y provocando por lo tanto que aparezca un mensaje indicando que “El empleado no existe”.

## Capítulo 6. Resultados Obtenidos

El resultado obtenido al concluir el desarrollo del sistema se muestra a continuación mediante algunas imágenes de como se ven las terminales ya en funcionamiento. Cada imagen se acompaña de su descripción correspondiente.

Cabe aclarar que muchas de las imágenes aquí mostradas pueden no ser idénticas a la vista real de las terminales tal como quedaron instaladas en la empresa, puesto que varios de los elementos que se muestran en cada pantalla de la Terminal se pueden personalizar.

Esto significa que los principales elementos de interfaz con el usuario como lo son el logo de la empresa y los textos de los mensajes, se pueden modificar en cualquier momento.

Al principio, cuando se ejecuta la aplicación, la Terminal obtiene su nombre de la tabla de configuración en su base de datos. Pero si éste no existe, lo solicita al usuario:



Figura 6.1 *Mensaje solicitando el nombre de la Terminal*

Después, la Terminal se comunica con el servidor para obtener la configuración que debe tomar, Solicita también que le sean enviados el logo que debe mostrar y los archivos de sonido que debe reproducir:



Figura 6.2 *Terminales obteniendo la configuración, el logo de la empresa y los archivos de sonido*

La Terminal entonces muestra el logo en pantalla y procede a activar el hardware que de acuerdo a la configuración se debe utilizar:



Figura 6.3 *Terminal preparando los elementos de hardware que utilizará*

Una vez que está lista para trabajar, muestra en pantalla todos los mensajes, tal como aparecen en la tabla de configuración, aplicando formato mediante la clase *CMensajes* cuando sea necesario:



Figura 6.4 *Terminal esperando a que se realice un proceso*

La Terminal continuará en ese estado mientras nadie haga uso de ella. Una vez transcurrido el tiempo establecido en la tabla de configuración, automáticamente comenzará a realizar procesos internos, entre los cuales se comunicará con el servidor para compartir información.

Aquí se muestran dos ejemplos de ello:



Figura 6.5 *Terminal realizando procesos internos*

En este caso, la lectora está informando que está actualizando el contenido de la tabla de personal de acuerdo a lo que existe en el servidor.



Figura 6.6 *Terminal realizando procesos internos*

En este otro caso, está informando que está enviando al servidor las huellas digitales que fueron capturadas en su módulo de huella digital y que por algún motivo no habían podido ser enviadas.

Cuando un empleado coloca su dedo en el sensor, el módulo realiza la identificación y si determina que éste no existe entonces la Terminal muestra un mensaje de “Acceso Denegado”, además de cualquier otra acción como reproducir un sonido o activar un relevador:



Figura 6.7 *Terminal denegando el acceso*

Supóngase que se trata de un empleado que sí debe poder ingresar, solo que aún no se han capturado sus huellas digitales. En tal caso, el personal de la



empresa encargado de administrar el sistema debe deslizar la credencial de control etiquetada como “AdminHD” para ingresar al modo de “Administración de Huellas Digitales”:



Figura 6.8 *Terminal ingresando al modo de Administración de Huellas Digitales*

En esta pantalla, se le solicita al empleado que coloque su huella digital o que deslice su credencial. La primera es para capturar de nuevo sus dos huellas digitales, la segunda es para verificar que sus huellas digitales fueron capturadas correctamente. Cuando el empleado desliza su credencial, aparece su número y su nombre, y se solicita que coloque su primera huella digital:



Figura 6.9 *Terminal solicitando al empleado que coloque su huella digital*

Al momento en que el empleado coloca su huella digital, ésta es capturada y se le muestra un mensaje en pantalla indicando la calidad con que se grabó:



Figura 6.10 Terminal mostrando la calidad con la que se capturó una huella digital

Exactamente lo mismo ocurre con la segunda huella digital del empleado. Una vez capturadas correctamente ambas huellas, éstas son enviadas al servidor, siempre y cuando haya comunicación disponible con el mismo, y la pantalla vuelve a su estado original:



Figura 6.11 Terminal en modo de Administración de Huellas Digitales

Ahora, si el empleado coloca su huella digital en el sensor, el módulo lo identificará satisfactoriamente y mostrará su nombre en pantalla:



Figura 6.12 *Terminal identificando a un usuario en el modo de Administración de Huellas Digitales*

Para salir, el administrador tiene que deslizar la credencial de control etiquetada como “Terminar”, con lo que la aplicación regresa a la pantalla principal:



Figura 6.13 *Terminal esperando a que se realice un proceso*

Esta vez, cuando el empleado coloca su dedo en el sensor, el sistema ya le da acceso:



Figura 6.14 *Terminal permitiendo el acceso a un empleado*

A continuación se muestran algunas fotografías reales del sistema en funcionamiento:

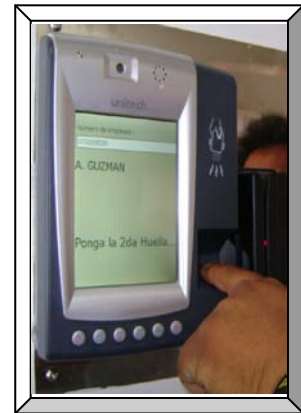
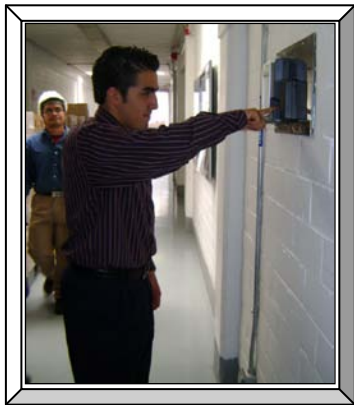


Figura 6.15 *Fotografías reales del sistema en funcionamiento*

## Capítulo 7. Consideraciones Técnico-Económicas

Las terminales son equipos MR650 de la marca Unitech® que cuentan con un módulo de huella digital integrado. Además cuentan con un lector de tarjetas de proximidad, un puerto serie RS232<sup>1</sup>, un puerto serie universal y un puerto Ethernet<sup>2</sup>. Tienen batería de respaldo con duración de hasta aproximadamente dos horas, tres relevadores para mandar pulsos eléctricos al exterior y cuatro entradas digitales para recibir instrucciones desde algún control externo. Tienen una cámara fotográfica integrada y también tienen micrófono y bocinas.

Su sistema operativo es Windows CE 5.0. No tienen disco duro real, solo tienen una memoria Flash de 64MB y tienen 128MB de memoria RAM. Se programan con los lenguajes de programación Visual Basic .NET, Visual C# .NET, Visual C++ Embedded o Visual Basic Embedded. Tienen la forma que se muestra en la figura 7.1.



Figura 7.1 Vista de una Terminal MR650 de Unitech®. Tomada de la página Web del fabricante

Cada una tiene un precio de \$3,300. En total las nueve terminales suman un monto de \$30,000.

Los lectores de código de barras son equipos MS146 también de la marca Unitech® con opción de comunicarse mediante puerto serie o puerto USB. En este caso se conectaron mediante puerto serie. Tienen la forma que se muestra en la figura 7.2.

---

<sup>1</sup> RS232 es un estándar de la industria para el puerto serie. Otro estándar es el RS485.

<sup>2</sup> Ethernet es un estándar de la industria para el puerto de comunicación en red.



Figura 7.2 Vista de un lector de código de barras MS146 de Unitech®. Tomada de la página Web del fabricante

Cada uno tiene un precio de \$150. En total los nueve lectores suman un monto de \$1,350.

Los módulos de huella digital son equipos SFM3500 de la marca Suprema® y vienen acompañados de sensores AF-S2 de la marca Authentec®, capaces de almacenar y reconocer hasta 9000 huellas en el modo “uno a muchos”. También se utilizó mediante puerto serie. Tienen la forma que se muestra en la figura 7.3.



Figura 7.3 Vista del módulo SFM3500 de Suprema®. Tomada de la página Web del fabricante

Cada uno tiene un precio de \$250. En total los nueve lectores suman un monto de \$2,250.

La inversión inicial que realizó la empresa al adquirir nueve terminales y nueve lectores de código de barras fue de \$31,350. Si la empresa hubiera optado por adquirir otras terminales probablemente de otra marca, que tuvieran las características necesarias para realizar adecuadamente el proceso de identificación del personal, hubiera tenido que gastar una cantidad similar.

Al adquirir solamente los módulos de huella digital el gasto fue de \$2,250, consiguiendo así un ahorro de casi \$30,000.

NOTA: Todos los precios aquí anotados son precios aproximados en dólares americanos. No incluyen gastos de envío ni gastos aduanales.
---

## Capítulo 8. Conclusiones

El sistema que se desarrolló funciona de la forma esperada, produciendo que la empresa ahorrara una fuerte cantidad de dinero al utilizar en un 90% o 95% el hardware que se adquirió originalmente, reemplazando solamente aquel que no soportaba la capacidad de procesamiento requerido por la empresa.

Esto no quiere decir que ese hardware se haya desechado, pues para aprovechar los módulos lectores de huella digital que venían instalados de fábrica dentro de las terminales, se les adaptó una fuente de alimentación y un conector de tipo DB9<sup>1</sup> para conectarlos al puerto serie de una computadora, y serán utilizados en escuelas para dar cursos a los estudiantes de la carrera de electrónica sobre cómo trabajan y cómo se controlan este tipo de módulos.

Esto permitirá que nuevas generaciones los conozcan y aprendan las bases sobre la identificación de personas mediante huella digital, lo que les ayudará para que en un futuro sean capaces de programar y controlar no solo esos módulos, sino cualquier módulo fabricado por cualquier empresa tan solo basándose en la hoja de datos del equipo.

Al igual que todo sistema de cómputo, cuando se realizó la instalación ya en la fábrica donde debía de funcionar se tuvieron algunos problemas que poco a poco se fueron resolviendo. A continuación se comentarán los problemas que mas trabajo costó resolver.

Al principio, poco antes de colocar las terminales en los lugares donde quedarían finalmente, se realizaron algunas pruebas sobre un escritorio obteniendo un tiempo promedio de identificación de las huellas digitales de 5.5 segundos, y en algunos casos la identificación demoraba hasta 10 segundos.

Para corregir este problema se realizaron muchas pruebas y se lograron detectar los motivos por los cuales la Terminal se tardaba tanto tiempo en responder:

- Se detectó que la Terminal es un poco lenta al momento actualizar los mensajes que se muestran en la pantalla, y estos se actualizaban constantemente aún cuando no hubieran cambiado. Al hacer que

---

<sup>1</sup> Se le llama DB9 al conector físico del puerto serie.

solamente se actualicen cuando sea necesario el tiempo de respuesta se redujo 1 segundo.

- Se ajustaron los parámetros de calibración del sensor de huella digital de adecuarlo a las condiciones en las que debería de trabajar. Esto redujo el tiempo medio segundo.
- Antes de mostrar el mensaje en la pantalla indicando al empleado que se le había permitido el acceso, primero se grababa el movimiento en la base de datos y se activaba el sonido relacionado con el acceso permitido. Al invertir el orden en el que se ejecutan estos procesos se redujo el tiempo medio segundo.
- Se reemplazó el sonido que se reproducía diciendo “Adelante”, para hacerlo más pequeño tanto en duración como en tamaño en disco. El tiempo se redujo medio segundo.

Después de aplicar todas estas correcciones el tiempo de respuesta de las terminales se quedó en 3 segundos.

Para los casos en los que el reconocimiento se demoraba demasiado tiempo (hasta 10 segundos en algunos casos), se detectó que las personas no colocaban bien su dedo en la superficie del lector, lo cual hacía lento el proceso pues el sensor se tardaba en darse cuenta de que habían colocado una huella. Cuando se le enseñó a los empleados a colocar los dedos correctamente, el tiempo de identificación con ellos también llegó a 3 segundos.

También hubo algunos problemas con respecto a la instalación eléctrica en los que se tuvo que solicitar el apoyo al personal de mantenimiento para que realizara las modificaciones necesarias. A continuación se enlistan los más importantes:

- Algunos elementos eléctricos como tomacorrientes y cables estaban dañados tal como se muestra en la figura 8.1, produciendo falsos en la alimentación de algunas terminales.

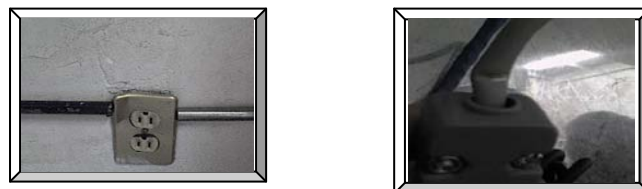


Figura 8.1 *Ejemplo de elementos eléctricos dañados*



- Algunas terminales las colocaron con un sobrante excesivamente largo de cable de red, lo cual da lugar a posibles fluctuaciones debidas a atenuaciones y/o distorsiones de la señal.

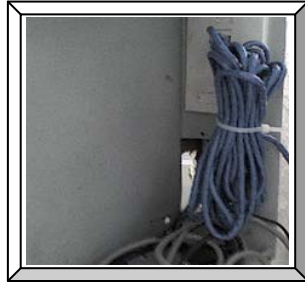


Figura 8.2 *Ejemplo de cables de red excesivamente largos*

Además de todo esto, la fábrica cuenta con una red eléctrica protegida que en casos de emergencia se alimenta por medio de una planta generadora de electricidad. En esa red eléctrica se conectan las maquinarias y los equipos críticos de la empresa.

Para que las terminales tampoco se apaguen fue necesario convencer al personal de la empresa que se encargaría de administrarlo para que mandaran conectar las terminales en dicha red.

Por ahora el sistema se encuentra instalado y funcionando en fase de pruebas, y se pretende que sea liberado en las próximas semanas, cuando la empresa determine que se han probado lo suficiente y ha llegado el momento de sacarles provecho.

## Anexos

### A. Rutinas del Servicio Web

A.1 Rutina del servicio Web para obtener los datos de una persona, los cuales serán grabados en alguna de las terminales.

```
<WebMethod()> _
Public Function RecibirPersonalDesdeElServidor(ByVal id As Long)
    As Personal

    Dim res As New Personal
    Dim query As String
    res.ExitoError = Estatus.EXITO

    Try
        Dim bd As New AccesoDatosSQL

        ' Lee los datos de la persona
        query = "select per.udent, per.codBarras, per.proximidad,
                    per.nombre, per.nacimiento, per.ingreso, per.baja "
        query &= "from personal per inner join cambiosPersonal camb
                    on per.udent = camb.udent "
        query &= "where camb.id = " & id

        If bd.LeerRegistrosDataset(query) = Estatus.ERROR Then
            Throw New Exception(bd.errorDesc)
        End If

        ' Llena la estructura de Personal
        With bd.dataSet.Tables(0).Rows(0)
            res.udent = .Item("udent")
            res.codBarras = .Item("codBarras")
            res.proximidad = .Item("proximidad")
            res.nombre = .Item("nombre")
            res.nacimiento = .Item("nacimiento")
            res.ingreso = .Item("ingreso")
            res.baja = .Item("baja")
        End With

        ' Asigna el estatus de transmitido al registro
        query = "update cambiosPersonal set enviado = 1
                    where id = " & id

        If bd.EjecutarUpdate(query) = Estatus.ERROR Then
            Throw New Exception(bd.errorDesc)
        End If

        Catch ex As Exception
            res.ExitoError = Estatus.ERROR
            res.errorDesc = ex.Message
        End Try

        Return res

    End Function
```

A.2 Rutina del servicio Web para obtener una huella digital que se encuentra almacenada en el servidor y que debe grabarse en alguna de las terminales.

```
<WebMethod()> _
Public Function RecibirHuellaDesdeElServidor(ByVal lectora As String,
                                             ByVal id As Integer, ByVal index As Byte) As Byte()

    Dim fs As IO.FileStream = Nothing
    Dim bd As New AccesoDatosSQL()
    Dim datosArchivo(0) As Byte
    Dim query As String

    Try

        ' Lee del archivo de disco
        If Not IO.File.Exists(Server.MapPath(Dir.HUELLAS_DIR &
                                             id & "_" & index & ".hue")) Then
            Throw New IO.FileNotFoundException()
        End If

        fs = New IO.FileStream(Server.MapPath(Dir.HUELLAS_DIR &
                                             id & "_" & index & ".hue"), IO.FileMode.Open)
        ReDim datosArchivo(fs.Length - 1)
        fs.Read(datosArchivo, 0, fs.Length)

        ' Actualiza la tabla para indicar que la huella digital ya ha
                                                sido enviada
        query = "update cambiosPersonal set enviadoHuella" &
               index & " = 1 " & "where uident = " & id &
               " and lectora = " & comillas(lectora)

        If bd.EjecutarUpdate(query) = Estatus.ERROR Then
            Throw New Exception(bd.errorDesc)
        End If

    Catch ex As IO.FileNotFoundException
        ' La huella no existe
        ReDim datosArchivo(0)
        datosArchivo(0) = 1

    Catch ex As Exception
        ' Error al leer del archivo
        ReDim datosArchivo(0)
        datosArchivo(0) = 0

    Finally
        If fs IsNot Nothing Then
            fs.Close()
        End If

    End Try

    Return datosArchivo

End Function
```

A.3 Rutina del servicio Web para obtener un listado de los empleados de nuevo ingreso, de los que se les ha modificado algún dato y de los que se han dado de baja.

```
<WebMethod()> _
Public Function RecibirCambiosPersonal(ByVal lectora As String)
                                                As String

    Dim cad As String = ""
    Dim query As String

    Try

        Dim bd As New AccesoDatosSQL

        ' Lee los cambios que hubo en el personal
        query = "select id from cambiosPersonal where enviado = 0 and
                lectora = " & comillas(lectora) & " order by fecha"

        If bd.LeerRegistrosDataset(query) = Estatus.ERROR Then
            Throw New Exception(bd.errorDesc)
        End If

        ' Manda la lista de cambios en el personal
        ' separados por comas
        Dim i As Integer

        For i = 0 To bd.dataSet.Tables(0).Rows.Count - 1
            cad &= Trim(Str(bd.dataSet.Tables(0).Rows(i).Item(0))) &
                                                                _COMA
        Next

    Catch ex As Exception

        cad = ""

    End Try

    Return cad

End Function
```

## B. Rutinas de la Aplicación de las Terminales

### B.1 Rutina de la clase *CCamara* para tomar una fotografía.

```
Public Sub TomarFoto(ByVal uident As Integer,
                    Optional ByVal x As Short = 10,
                    Optional ByVal y As Short = 10,
                    Optional ByVal ancho As Short = 100,
                    Optional ByVal largo As Short = 100)

    ' Vacio significa que no tomo foto
    archivo = ""

    If Config.sacarFotos AndAlso estatusHardware = Activo Then
        If (Now.Hour >= Config.horaInicioFotos1 And Now.Hour <
            Config.horaFinFotos1) Or _
            (Now.Hour >= Config.horaInicioFotos2 And Now.Hour <
            Config.horaFinFotos2) Then

            Try
                ' Toma una foto y la guarda en disco con formato:
                '          aaaammdd_hhmmss_uident
                archivo = FechaHora_Texto(Now,
                    FormatoFechas.aaaammdd, "", "", "_")
                    & "_" & uident & ".jpg"

                ActivarPreview(x, y, ancho, largo)
                CaptureImage(Dir.FOTOS_DIR & archivo, ancho, largo)
                DesactivarPreview()

            Catch ex As Exception
                archivo = ""
            End Try

        End If
    End If
End Sub
```

### B.2 Rutina de la clase *ConexionWS* para enviar una huella digital al servidor.

```
Public Function EnviarHuellaAlServidor(ByVal id As Integer,
                                       ByVal index As Byte) As Estatus

    Dim res As Estatus = Estatus.EXITO
    Dim nombreArchivo As String
    Dim fs As IO.FileStream = Nothing
    Dim datosArchivo As Byte()

    Try
        ' Lee del archivo de disco
        nombreArchivo = Dir.HUELLAS_DIR & id & "_" & index & ".hue"
        fs = New IO.FileStream(nombreArchivo, IO.FileMode.Open)
        ReDim datosArchivo(fs.Length - 1)
        fs.Read(datosArchivo, 0, fs.Length)
    End Try
End Function
```

```

        fs.Close()
        fs = Nothing

        ' Hace el envio
        If ws.EnviarHuellaAlServidor(objLectora.lectora, id, index,
                                    datosArchivo) = Estatus.ERROR Then
            Throw New Exception("Falla en el Webservice")
        End If

    Catch ex As Exception
        ' Error al enviar el archivo
        res = Estatus.ERROR
        If fs IsNot Nothing Then
            fs.Close()
        End If
    End Try

    Return res
End Function

```

### B.3 Rutina de la clase *ConexionWS* para enviar un movimiento al servidor.

```

Public Function EnviarMovimientosAlServidor() As Estatus
    Dim res As Estatus = Estatus.EXITO
    Dim movto As New Movimiento

    Try
        ' Obtiene el movto
        With bd.dataSet.Tables(0).Rows(CMovimientos.i)
            movto.uiden = .Item("uiden")
            movto.fechahora = .Item("fecha")
            movto.lectora = .Item("lectora")
            movto.movto = .Item("movto")
            movto.comenta = .Item("comenta")
            movto.foto = .Item("foto")
            movto.identificador = .Item("identificador")
        End With

        ' Lo envia al servidor
        If ws.EnviarMovimientoAlServidor(movto) = Estatus.ERROR Then
            Throw New Exception("Falla en el Webservice.")
        End If

    Catch ex As Exception
        res = Estatus.ERROR
    End Try

    Return res
End Function

```

B.4 Rutina de la clase *HD\_SFM3500* para grabar en el módulo una huella digital que se encuentre en un archivo de disco.

```
Private Sub enrollByTemplate(ByVal id As Integer, ByVal index As Byte)

    _estatus = EstatusLector.Ocupado

    ' Captura una huella
    comando = ENROLL_BY_TEMPLATE
    pac.Command = ENROLL_BY_TEMPLATE
    Me.setParam(id)
    Me.setSize(_TAMAÑO_TEMPLATE)
    pac.Flag_Error = _ADD_NEW

    ' Prepara la huella para mandarla
    Dim datosArchivo(_TAMAÑO_TEMPLATE) As Byte

    Try
        Dim fs As New IO.FileStream(Dir.HUELLAS_DIR & id & "_" &
            index & ".hue", IO.FileMode.Open, IO.FileAccess.Read)
        fs.Read(datosArchivo, 0, _TAMAÑO_TEMPLATE)
        fs.Close()
        datosArchivo(_TAMAÑO_TEMPLATE) = END_CODE ' END_CODE

    Catch ex As Exception
        errordesc = ex.Message
    End Try

    ' Lo manda todo
    puerto.Write(Me.getPacketBytes, 0, _TAMAÑO_PACKET)
    puerto.Write(datosArchivo, 0, _TAMAÑO_TEMPLATE + 1)

End Sub
```

B.5 Caso especial de la clase *HD\_SFM3500* en el cual el módulo envía más información que un solo paquete de datos. Se utiliza cuando se le solicita al módulo una huella digital para ser grabada en un archivo de disco.

```
Case READ_TEMPLATE
    Select Case pac.Flag_Error
        Case SUCCESS ' Solo habia una
            ' Espera los bytes
            Me.esperarAQueLleguenLosBytes(_TAMAÑO_TEMPLATE + 1)
            If errorBytes Then GoTo xFin

            ' Recibe los bytes y graba la huella
            resp = LeerYGrabarLosBytesDeUnaHuella(0)

        Case [CONTINUE]
            ' Espera los bytes
            Me.esperarAQueLleguenLosBytes(_TAMAÑO_TEMPLATE + 1)
            If errorBytes Then GoTo xFin
```

```

' Recibe los bytes y graba la huella
resp = LeerYGrabarLosBytesDeUnaHuella(0)

' Recibe el segundo paquete'''
' Espera los bytes
Me.esperarAQueLleguenLosBytes(_TAMAÑO_PACKET)
If errorBytes Then GoTo xFin

puerto.Read(datos, 0, _TAMAÑO_PACKET)
Me.setPacketBytes(datos)

Select Case pac.Flag_Error
Case SUCCESS
    ' Lee la Segunda Huella'''
    ' Espera los bytes
    Me.esperarAQueLleguenLosBytes(
        _TAMAÑO_TEMPLATE + 1)
    If errorBytes Then GoTo xFin

    ' Recibe los bytes y graba la huella
    resp = LeerYGrabarLosBytesDeUnaHuella(1)

Case [CONTINUE]
    ' Lee la Segunda Huella'''
    ' Espera los bytes
    Me.esperarAQueLleguenLosBytes(
        _TAMAÑO_TEMPLATE + 1)
    If errorBytes Then GoTo xFin

    ' Recibe los bytes y graba la huella
    resp = LeerYGrabarLosBytesDeUnaHuella(1)

    ' Ignora cualquier otra huella
    Me.VaciarBufferDeRecepcion()

Case Else ' Error
    resp = Estatus.ERROR

End Select

Case Else ' Error
    resp = Estatus.ERROR

End Select

```

B.6 Rutina de la clase *HD\_SFM3500* para leer los datos recibidos en el puerto serie y grabarlos en un archivo de disco.

```

Private Function LeerYGrabarLosBytesDeUnaHuella(
    ByVal numHuella As Byte) As Estatus

    Dim ret As Estatus = EXITO
    Dim datosArchivo(_TAMAÑO_TEMPLATE - 1) As Byte
    Dim fs As IO.FileStream = Nothing

```



```

Try
    'Lee la huella y la guarda en disco
    puerto.Read(datosArchivo, 0, _TAMAÑO_TEMPLATE)
    fs = New IO.FileStream(Dir.HUELLAS_DIR & Me.id & "_" &
                           numHuella & ".hue", IO.FileMode.Create,
                           IO.FileAccess.Write)
    fs.Write(datosArchivo, 0, _TAMAÑO_TEMPLATE)
    fs.Close()

Catch ex As Exception
    errorDesc = ex.Message
    ret = [ERROR]
Finally

    If fs IsNot Nothing Then
        fs.Close()
    End If

End Try

' Lee el END_CODE
puerto.ReadByte()

Return ret

End Function

```

B.7 Código de la clase *Hardware* encargada de activar y desactivar el hardware de las terminales.

```

' Huella Digital
If Config.usaHuellas Then
    If lectorHuella.estatusHardware = Inactivo Then

        Select Case sensor
            Case SensorHuellaDigital.BioScript_MV1510
                lectorHuella.paramSecurityLevel =
                    Config.HD_SecurityLevel
                lectorHuella.Activar()

            Case SensorHuellaDigital.Suprema_SFM3500
                lectorHuella.Activar()
                lectorHuella.setSystemParameter(
                    HD_SFM3500._FAST_MODE,
                    Config.HD_FastMode)
                lectorHuella.setSystemParameter(
                    HD_SFM3500._FREE_SCAN_DELAY,
                    Config.HD_FreeScanDelay)
                lectorHuella.setSystemParameter(
                    HD_SFM3500._LIGHTING_CONDITION,
                    Config.HD_LightingCondition)
                lectorHuella.setSystemParameter(
                    HD_SFM3500._ROTATION,

```

```

Config.HD_Rotation)
lectorHuella.setSystemParameter(
    HD_SFM3500._SENSITIVITY,
    Config.HD_Sensitivity)

End Select
End If

Else

    If lectorHuella.estatusHardware = Activo Then
        lectorHuella.Desactivar()
    End If

End If

' Relevadores
If Config.usaRelayPermitido Then
    If relPermitir.estatusHardware = Inactivo Then
        relPermitir.Activar()
    End If
Else
    If relPermitir.estatusHardware = Activo Then
        relPermitir.Desactivar()
    End If
End If

If Config.usaRelayDenegado Then
    If relDenegar.estatusHardware = Inactivo Then
        relDenegar.Activar()
    End If
Else
    If relDenegar.estatusHardware = Activo Then
        relDenegar.Desactivar()
    End If
End If

' Camara
If Config.sacarFotos Then
    If camara.estatusHardware = Inactivo Then
        camara.Activar()
    End If
Else
    If camara.estatusHardware = Activo Then
        camara.Desactivar()
    End If
End If

' LectorCodigo de Barras (No se desactiva para poder usar las
' credenciales de control)
If lectorCodBar.estatusHardware = Inactivo Then
    lectorCodBar.Activar()
End If

If nCodBarras = CMIRG.Terminar Then
    If lectorCodBar.estatusHardware = Activo Then
        lectorCodBar.Desactivar()
    End If

```

```
End If
```

B.8 Código de la clase *Config* para obtener la configuración de la Terminal mediante el servicio Web y grabarla en la base de datos.

```
' Obtiene la configuracion
If ws.RecibirConfiguracionDesdeElServidor() = Estatus.ERROR
    Then
        Throw New Exception("Error al obtener la configuración desde
                                el servidor.")
End If

' Graba la nueva configuracion
bd.query = "update configLect set "
bd.query &= "tipo = " & tipoLectora & _COMA
bd.query &= "usaBarras = " & IIf(usaBarras, 1, 0) & _COMA
bd.query &= "usaHuellas = " & IIf(usaHuellas, 1, 0) & _COMA
bd.query &= "usaProxim = " & IIf(usaProximidad, 1, 0) & _COMA
bd.query &= "sacarFotos = " & IIf(sacarFotos, 1, 0) & _COMA
bd.query &= "desayunos = " & cantidadDesayunos & _COMA
bd.query &= "comidas = " & cantidadComidas & _COMA
bd.query &= "relayPermitido = " & IIf(usaRelayPermitido, 1, 0) & _COMA
bd.query &= "relayDenegado = " & IIf(usaRelayDenegado, 1, 0) & _COMA
bd.query &= "mensajeFijo = " & comillas(mensajeFijo) & _COMA
bd.query &= "mensajePermitido = " & comillas(mensajePermitido) & _COMA
bd.query &= "mensajeDenegado = " & comillas(mensajeDenegado) & _COMA
bd.query &= "horaInicioWS = " & horaInicioWS & _COMA
bd.query &= "horaFinWS = " & horaFinWS & _COMA
bd.query &= "hrIniDesayuno = " & horaInicioDesayuno & _COMA
bd.query &= "hrFinDesayuno = " & horaFinDesayuno & _COMA
bd.query &= "empresa = " & empresa & _COMA
bd.query &= "Hr1InicioFotos = " & horaInicioFotos1 & _COMA
bd.query &= "Hr1FinFotos = " & horaFinFotos1 & _COMA
bd.query &= "Hr2InicioFotos = " & horaInicioFotos2 & _COMA
bd.query &= "Hr2FinFotos = " & horaFinFotos2 & _COMA
bd.query &= "segundosSinActividad = " & segundosSinActividad & _COMA
bd.query &= "cantidadHuellas = " & cantidadHuellas & _COMA
bd.query &= "administradores = " & comillas(administradores)

If bd.EjecutarComando() = Estatus.ERROR Then
    Throw New Exception(bd.errorDesc)
End If
```

B.9 Rutina de la clase *Config* para obtener el archivo de configuración del módulo de huella digital, el cual se encuentra en el servidor.

```
Public Function RecibirArchivoConfigHuellaDigitalDesdeElServidor()
    As Estatus

    Dim fs As IO.FileStream = Nothing
    Dim datosArchivo() As Byte
```

```

Dim res As Estatus = Estatus.EXITO

Try
    ' Lo recibe
    datosArchivo =
        ws.RecibirArchivoConfigHuellaDigitalDesdeElServidor()
    If datosArchivo.Length = 1 Then
        Throw New Exception("Falla en el WebService")
    End If

    ' Lo guarda en disco
    fs = New IO.FileStream(Arch.ARCH_CONFIG_HUELLA_DIGITAL,
                           IO.FileMode.Create)

    fs.Write(datosArchivo, 0, datosArchivo.Length)
    fs.Close()
    fs = Nothing

Catch ex As Exception
    ' Error al enviar el archivo
    If fs IsNot Nothing Then
        fs.Close()
    End If
    res = Estatus.ERROR
End Try

Return res

End Function

```

## B.10 Código de la pantalla *frmInicio* para ejecutar las rutinas del servicio Web encargadas de enviar y recibir información.

```

Select Case objLectora.siguienteComunicacion
Case ComunicacionLectora.Inicio
    ' Inicio de la comunicacion con el web service
    CMensajes.AsignarMensajePequeño(Me, "Inicio comunicacion ws")
    objLectora.siguienteComunicacion += 1

Case ComunicacionLectora.Recibir_CambiosPersonal
    CMensajes.AsignarMensajePequeño(Me, "Obteniendo cambios
                                     personal ws")

    CPersonal.RecibirCambiosPersonal()
    objLectora.siguienteComunicacion += 1

Case ComunicacionLectora.Recibir_Personal
    CMensajes.AsignarMensajePequeño(Me, "Obteniendo personal ws")
    objLectora.terminoComunicacion = False
    ' Si ya no hay cambios que bajar, termina este estado de
                                     comunicacion

    If CPersonal.Ids.Length = 0 Then
        objLectora.terminoComunicacion = True
    Else
        'Si ocurrio un error, pudo ser x que se desconecto el
        'cable. Asi que termina con este estado de comunicacion
        If CPersonal.RecibirPersonalDesdeElServidor() =

```

```

                                Estatus.ERROR Then
        objLectora.terminoComunicacion = True
    End If
End If

If objLectora.terminoComunicacion Then
    objLectora.siguienteComunicacion += 1
End If

Case ComunicacionLectora.Envia_Movimientos
    CMensajes.AsignarMensajePequeño(Me, "Enviando movimientos ws")
    objLectora.terminoComunicacion = False

    ' Si ocurrio un error, pudo ser x que se desconecto el cable.
    ' Asi que termina con este estado de comunicacion
    If CMovimientos.EnviaMovimientosAlServidor() = Estatus.ERROR
        Then
        objLectora.terminoComunicacion = True
    Else
        ' Si ya no hay movtos que subir, termina este estado de
        comunicacion
        If CMovimientos.CantidadMovtos = 0 Then
            objLectora.terminoComunicacion = True
        End If
    End If

    If objLectora.terminoComunicacion Then
        objLectora.siguienteComunicacion += 1
    End If

```

B.11 Código de la pantalla *frmAdminHD* donde se inicia la captura de las huellas digitales de un empleado.

```

Case EstatusAdmin.ConfirmandoNumero
    ' Credenciales de control
    If nCodBarras = CMIRG.Terminar Then Exit Sub

    If nCodBarras = CMIRG.AdminHD Then
        objLectora.estadoAdmin = EstatusAdmin.EsperandoCredencial
        Exit Sub
    End If

    Me.txtEmpleado.Text = nCodBarras
    CMensajes.AsignarMensajeAdmin(Me, "Procesando...", False)

    ' Borra las huellas de ese empleado para capturarlas de nuevo
    If CVerificador.ExisteEmpleado(nCodBarras) Then
        lblNombre.Text = CVerificador.nombre
        Id = CVerificador.udent
        lectorHuella.DeleteID(Id)
        objLectora.estadoAdmin = EstatusAdmin.EsperandoHuella1
    Else
        CMensajes.AsignarMensajeAdmin(Me, "El empleado no existe")
        objLectora.estadoAdmin = EstatusAdmin.EsperandoCredencial
    End If

```

```
End If
```

B.12 Código de la pantalla *frmAdminHD* donde se captura la primera huella digital de un empleado.

```
Case EstatusAdmin.EsperandoHuella1
    CMensajes.AsignarMensajeAdmin(Me, "Ponga la 1er huella...", False)

    If lectorHuella.Enroll(Id) = Estatus.EXITO Then

        ' Actualiza el registro
        bd.query = "update personal set huella = 1 where uident=" & Id
        bd.EjecutarComando()
        MsgBox.WaitWindow("Huella digital grabada con éxito:
                           Quality = " & lectorHuella.quality, 2000)

        If Config.cantidadHuellas = 1 Then
            objLectora.estadoAdmin = EstatusAdmin.EnvianadoHuellas
        Else
            objLectora.estadoAdmin = EstatusAdmin.EsperandoHuella2
        End If
    Else
        CMensajes.AsignarMensajeAdmin(Me, "Error al obtener huella")
    End If
```

B.13 Código de la pantalla *frmAdminHD* donde se envían al servidor las huellas capturadas.

```
Case EstatusAdmin.EnvianadoHuellas
    CMensajes.AsignarMensajeAdmin(Me, "Envianado Huellas...", False)

    ' Sacar las huellas del modulo
    Select Case sensor
        Case SensorHuellaDigital.BioScript_MV1510
            lectorHuella.Copiar_Modulo_Lectora(Id, 0)
            If Config.cantidadHuellas = 2 Then
                lectorHuella.Copiar_Modulo_Lectora(Id, 1)
            End If

        Case SensorHuellaDigital.Suprema_SFM3500
            lectorHuella.Copiar_Modulo_Lectora(Id, 0)

    End Select

    ' Envia primer huella
    If ws.EnviaHuellaAlServidor(Id, 0) = Estatus.ERROR Then
        CMensajes.AsignarMensajeAdmin(Me, "No hay comunicación con el
                                           servidor...", False)
        objLectora.estadoAdmin = EstatusAdmin.EsperandoCredencial
        Exit Sub
    Else
        IO.File.Delete(Dir.HUELLAS_DIR & Id & "_0.hue")
```

```

End If

If Config.cantidadHuellas = 2 Then
    ' Envía segunda huella
    If ws.EnvíarHuellaAlServidor(Id, 1) = Estatus.ERROR Then

        CMensajes.AsígnarMensajeAdmin(Me, "No hay comunicación
                                         con el servidor...", False)
        objLectora.estadoAdmin=EstatusAdmin.EsperandoCredencial
        Exit Sub
    Else
        IO.File.Delete(Dir.HUELLAS_DIR & Id & "_1.hue")
    End If

End If

objLectora.estadoAdmin = EstatusAdmin.EsperandoCredencial

```

B.14 Código de la pantalla *frmAdminHD* donde se considera el caso en el que el empleado coloca su huella digital sin antes haber deslizado su credencial.

```

' Pusieron el dedo sin pasar credencial
If Not nHuellaId = "" And objLectora.estadoAdmin =
                                EstatusAdmin.EsperandoCredencial Then

    ' Es para practica
    If CVerificador.ExisteEmpleado(, , nHuellaId) Then
        lblNombre.Text = CVerificador.nombre
        Me.txtEmpleado.Text = CVerificador.uidént
        CMensajes.AsígnarMensajeAdmin(Me, "")
        sonidos.Bien()
    Else
        CMensajes.AsígnarMensajeAdmin(Me,"El empleado no existe")
        sonidos.Mal()
    End If

    Delay(2000)

    ' Mensaje original
    If Config.empresa = Empresas.Buhler Then
        CMensajes.AsígnarMensajeAdmin(Me,"Teclee No. de Empleado...",
                                         False)
    Else
        CMensajes.AsígnarMensajeAdmin(Me, "Pase Credencial...",False)
    End If

    Me.txtEmpleado.Text = ""
    lblNombre.Text = ""
End If

```

## **Referencias Bibliográficas**

### **Libros:**

Superutilidades para Visual Basic .NET, 1ª Edición, 2003

Autor: Kris Jamsa

Editorial: McGraw-Hill

El Lenguaje Unificado de Modelado, 1ª Edición, 2001

Autor: Grady Booch

Editorial: Pearson

Object Thinking, 1ª Edición, 2004

Autor: David West

Editorial: Microsoft Press

Intelligent Biometric Techniques in Fingerprint and Face Recognition, 1ª Edición, 1999

Autores: L. C. Jain, U. Halici, I. Hayashi, S. B. Lee, S. Tsutsui

Editorial: CRC

### **Páginas Web**

La Enciclopedia Libre

<http://www.wikipedia.org>

Página Web de la Empresa Unitech®

<http://www.ute.com>

Página Web de la Empresa Suprema®

<http://www.supremainc.com>

Página Web de la Empresa Bioscrypt®

<http://www.bioscrypt.com>

Las marcas House Of Fuller, Bioscrypt, BioEnable, Suprema, Chinazhiwen, Biometrix, Unitech y Authentec, así como AF-S2, FIM10, MV1250, SFM3500, BioAccess v2, NAC3000, S3K LAN, MR650, MS146, M4 Terminal, Visual Basic .NET, Windows CE, SQL Server Mobile, SQL Server, Visual FoxPro, Visual C# .NET, Visual C++ Embedded, Visual Basic Embedded, Secugen Hamster, FINGER-007 y Bio-OFFICE OA100 son marcas registradas.